WBEM

Olivier Festor, Nizar Ben Youssef

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

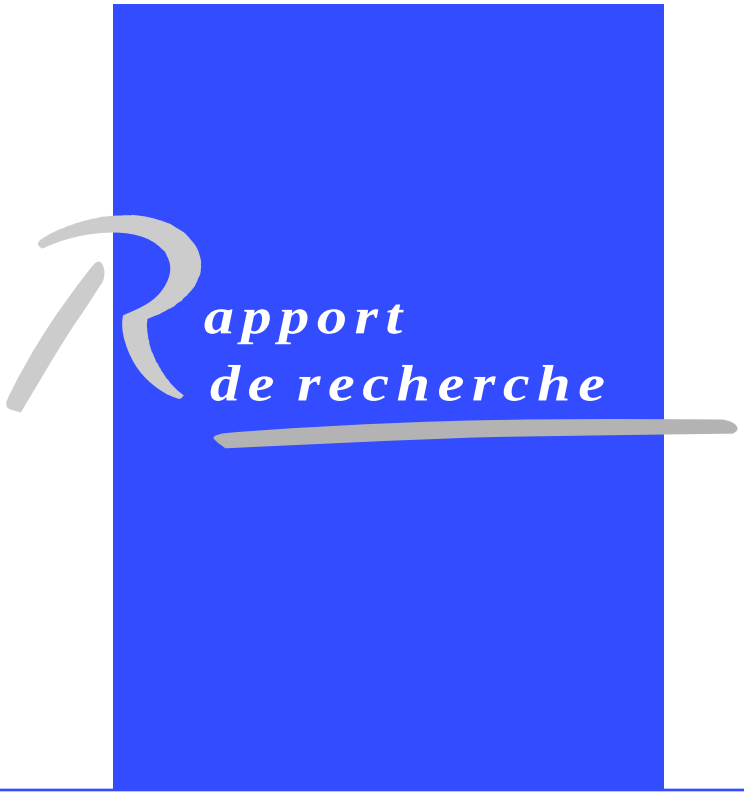
WBEM

Olivier Festor, Nizar Ben Youssef

No 3927

21 avril 2000

THÈME 1



ISSN 0249-6399 ISRN INRIA/RR--3927--FR

WBEM

Olivier Festor, Nizar Ben Youssef

Thème 1 — Réseaux et systèmes
Projet RÉSÈDAS

Rapport de recherche n 3927 — 21 avril 2000 — pages

Résumé : Ce rapport présente l'approche Web-Based Enterprise Management (WBEM) définie par le Distributed Management Task Force (DMTF). Il propose une introduction générale à l'approche visant à fournir aux non spécialistes de la supervision une vision globale des objectifs et capacités de l'approche. Il s'adresse également à des ingénieurs et chercheurs confirmés en supervision de réseaux et leur fournit une description détaillée de tous les composants et motivant ses intérêts et limites.

Mots-clé : CIM, HTTP, MOF, Supervision d'applications, Supervision de réseau, WBEM, XML

(Abstract: pto)

Ce travail a été partiellement réalisé dans le cadre de l'action ANTARES (Architectures et Nouvelles Technologies pour l'Administration des Réseaux et Service) II du GIE DYADE.

Unité de recherche INRIA Lorraine
Technopôle de Nancy-Brabois, Campus scientifique,
615 rue de Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY (France)
Téléphone : 03 83 59 30 30 - International : +33 3 3 83 59 30 30
Télécopie : 03 83 27 83 19 - International : +33 3 83 27 83 19
Antenne de Metz, technopôle de Metz 2000, 4 rue Marconi, 55070 METZ

WBEM

Abstract: This report is a tutorial for the Web-Based Enterprise Management (WBEM) approach initiated by the Distributed Management Task Force (DMTF) consortium. The tutorial provides a general introduction to this approach allowing people which are not confident with the management framework to get a broad view of the objectives and features of WBEM. The tutorial is also intended to be used by confirmedd network management engineers and researchers. To them, it provides an indepth description of all components which helps evaluating its strenght and limits.

Key-words: Applications Management CIM, HTTP, MOF, Network Management, WBEM, XML

Table des matières

1 Introduction

Apparue en juillet 1996, l'initiative Web-Based Enterprise Management (WBEM) a rapidement pris de l'ampleur dans le monde de la gestion des réseaux et des systèmes. C'est une initiative lancée par un ensemble d'entreprises dont les plus ardents défenseurs sont BMC Software Inc., Cisco Systems Inc., Compaq Computer Corp., IBM via Tivoli Systems, Intel Corp. et Microsoft Corp. Depuis Juin 1998, l'effort de normalisation de l'initiative a été pris en charge par le Distributed Management Task Force[1] (DMTF), consortium regroupant les industriels concernés par l'approche. Actuellement, de nombreuses entreprises, parmi lesquelles on retrouve les principaux fournisseurs de plate-formes de gestion, ont annoncé un support logiciel pour cette intiative et le nombre d'offres commerciales pour cette technologie ne cesse de croître.

WBEM a pour objectif principal de fournir un cadre unificateur pour la gestion des réseaux, services et applications au niveau de l'entreprise. Cela se traduit par la définition d'une architecture assurant l'interopérabilité entre applications au travers de la portabilité des informations de gestion indépendamment de leur origine. Les points suivants sont pris en compte :

- la création d'une vue homogène de l'ensemble des ressources gérées et ce quelque soit leur nature, localisation et/ou méthode d'accès ;
- la préservation de l'existant via une intégration des modèles de l'information, architectures de gestion et protocoles déjà déployés ;
- la constitution d'un support d'échange d'informations de gestion entre applications.

Ce dernier point vise principalement l'interopérabilité et le couplage d'applications de gestion, domaine qui n'était que peu abordé dans les autres approches de gestion.

Officiellement, WBEM n'a pas pour but de remplacer les technologies actuelles telles que Simple Network Management Protocol (SNMP) [4], Common Management Information Service (CMIS) [21] ou Desktop Management Interface (DMI) [5][2]. Si cela était le cas, WBEM serait condamnée avant même la mise sur le marché de plates-formes logicielles l'implémentant en raison de l'énorme dissémination de ces approches dans les réseaux et systèmes actuels. Le concept défendu est plutôt la mise en place d'un chapeau qui présente toutes les informations de gestion de manière uniforme aux applications[3].

Pour atteindre cet objectif, les quatres composants de base en gestion de réseaux ont été redéfinis. Ces composants sont :

- une architecture fonctionnelle définissant les intervenants dans la gestion, leur répartition, les fonctions qu'ils assurent et leurs rôles respectifs ;
- une approche de modélisation et un formalisme pour la spécification de nouveaux modèles de l'information (modèle informationnel) ;
- un modèle de l'information unificateur comportant un ensemble d'objets gérés de base définissant un modèle générique (modèle fonctionnel) ;
- un service et un protocole assurant la communication entre les entités (modèle de communication).

Dans la suite de ce chapitre, nous présentons en détails ces composants. Dans un premier temps, nous détaillons l'architecture proposée (section 2). Dans la section 3, nous étudions la composante modèle de l'information de l'approche. Cette étude est complétée dans les sections 4 et 5 dans lesquelles nous détaillons le langage de spécification et les modèles de l'information génériques définis dans l'approche. La section 6 présente le modèle de communication. Celui-ci est instancié dans la section 7 au travers de l'encapsulation XML/HTTP des requêtes WBEM.

L'intégration d'autres approches de gestion dans l'architecture WBEM est considérée dans la section 8. Les implémentations actuelles de WBEM sont presentées dans la section 9. Le chapitre est conclu par un inventaire des avantages et limites de l'approche suivi d'une conclusion (section 10).

1. Initialement connu sous le nom de Desktop Management Task Force, le DMTF a changé de nom depuis Mai 1999 afin de suivre l'évolution de WBEM qui se positionne de plus en plus sur le domaine de la gestion des réseaux et surtout des services!

2. `http://www.dmtf.org/spec/spec.html`

3. Ce qui revient en fait à considérer les autres approches comme du *legacy* et à proposer leur abandon en encourageant les développeurs à implémenter leurs nouveaux agents directement en WBEM.

2 L'architecture fonctionnelle

Assez classiquement, toute nouvelle approche de gestion définit sa propre terminologie pour classifier les acteurs dans le processus de gestion. Il s'agit principalement de spécifier les intervenants et de leur associer des rôles. Ces rôles vont définir les capacités d'une entité dans une interaction de gestion. WBEM ne déroge pas à la règle et définit sa propre terminologie. Celle-ci est présentée en détail dans cette section.

2.1 Entités et rôles

L'environnement de gestion, i.e. le monde des applications de gestion et des éléments gérés, est décomposé en un ensemble d'entités de gestion capables de coopérer[4].

2.1.1 Rôles

À chaque entité, on peut associer un ou plusieur rôles. Ces rôles sont schématisés dans la figure 1 :

- **client:** une entité prenant le rôle de client peut demander des informations de gestion ou l'exécution d'opérations de gestion à une entité dans le rôle serveur ;
- **serveur:** une entité remplissant le rôle serveur va maintenir des informations de gestion au sein d'une base d'informations de gestion et exécuter les opérations de gestion émises par des entités clientes ;
- **producteur:** une entité peut prendre un rôle de producteur si celle-ci peut émettre des indications vers des entités dans le rôle de consommateur. Est donc classifié comme producteur tout élément dans l'architecture de gestion qui implémente le service d'émission de notifications ;
- **consommateur:** une entité dans le rôle consommateur peut recevoir et traiter des indications émises par une ou plusieurs entités dans le rôle producteur.

L'affectation de rôle ou plutôt la classification d'une entité se fait via le support des services de communication associés à chaque rôle.

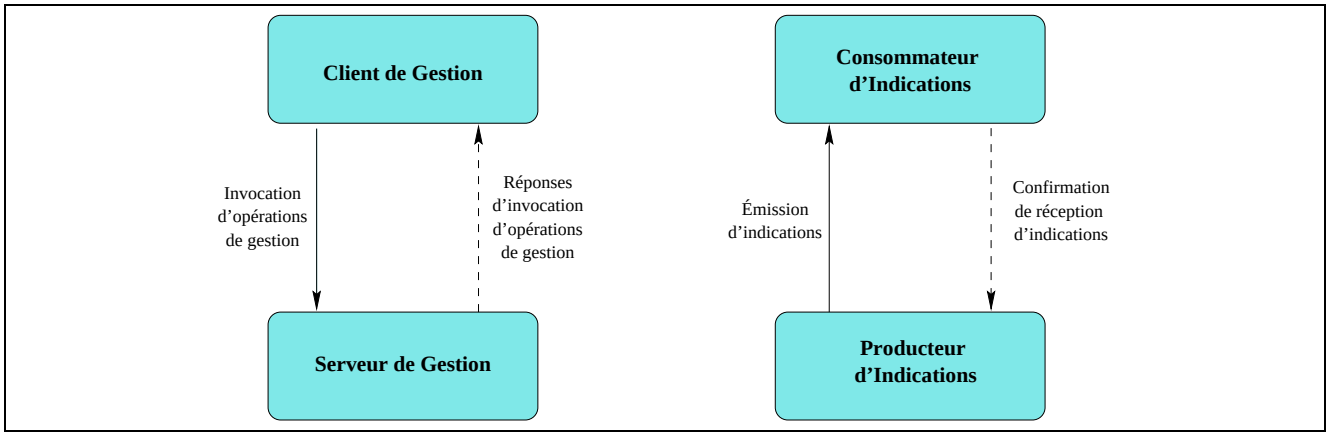


FIG. 1 – *Les rôles des intervenants dans l'architecture WBEM*

Dans un système de gestion déployé sur un réseau, une entité peut prendre plusieurs rôles. De plus, cette affectation de rôle n'est pas figée dans le temps. Par exemple, une entité peut à la fois être dans un rôle client face à un ou plusieurs serveurs d'informations de gestion et consommateur d'événements provenant d'un ou plusieurs producteurs de notifications. Cette même entité peut à un moment, décider de ne plus vouloir recevoir d'indications et va abandonner le rôle de consommateur. De même, un serveur d'information de gestion peut également être producteur de notifications ou d'indications.

Si l'on se ramène à la classification OSI des intervenants de la gestion de réseaux, un serveur et/ou un producteur d'indications serait défini comme un agent de gestion. Un client ainsi qu'un consommateur seraient définis comme gestionnaires. On retouve donc dans WBEM, les rôles traditionnels des architectures de gestion qui forment un sous ensemble des configurations possibles dans WBEM.

4. Ceci est la définition officielle. Il faut prendre dans cette définition le terme coopération au sens le plus large possible, c'est à dire la capacité de s'échanger des informations.

2.1.2 Entités

Les entités coopérantes définies dans l'initiative WBEM sont au nombre de quatre. Elles sont schématisées dans la figure 2 :

- l'**application** de gestion : constituant l'interface entre le domaine géré et le gestionnaire, elle permet à ce dernier d'accéder aux informations de gestion maintenues dans le gestionnaire CIM (Common Information Model voir section 3);
- le **gestionnaire** d'objets **CIMOM** (*CIM Object Manager*) : composant principal de l'architecture WBEM, il a la charge de maintenir une base d'informations de gestion (MIB : Management Information Base) et d'offrir aux applications de gestion une interface d'accès à cette base. Dans la terminologie WBEM, l'appellation MIB au niveau du CIMOM est abandonnée au profit de *CIM Object Repository*, plus proche de la réalité que la notion classique de MIB gérée par un agent directement attaché aux ressources. Le rôle d'un CIMOM dans l'architecture WBEM consiste à donner au gestionnaire une vue homogène de l'ensemble des éléments déployés dans le domaine géré, et ce grâce à leur modélisation dans un langage commun : CIM;
- le **provider** : Il assure la liaison entre le CIMOM et les ressources qu'il supervise. Aussi important que le CIMOM dans l'architecture WBEM, il réalise l'intégration dans le modèle CIM des informations relatives aux ressources qu'il gére. Le *provider* présente alors deux interfaces : une interface CIM pour communiquer avec le CIMOM, et une interface `native` (SNMP, CMIP, DMI, etc ...) pour communiquer avec les ressources gérées ;
- l'**agent** ou ressource gérée : attaché à un *provider* WBEM, il permet d'exécuter les opérations de gestion requises par le gestionnaire.

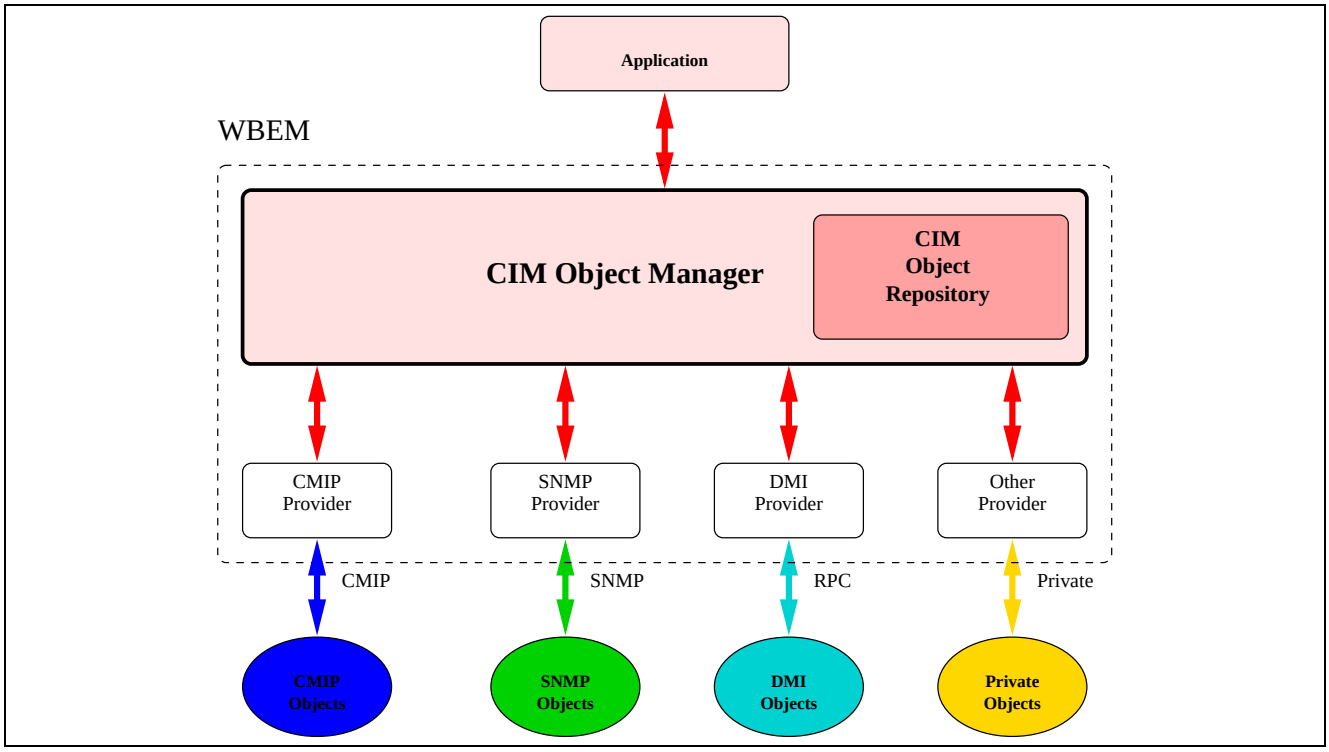


FIG. 2 – *Les entités dans l'architecture WBEM*

L'architecture présentée dans cette section, diffère[5] d'une architecture de gestion classique basée sur le paradigme de gestionnaire/agent. Elle définit par contre un cadre unificateur visant à assurer une gestion homogène d'un ensemble hétérogène de ressources, tout en conservant les structures et technologies existantes. En effet, grâce au couplage CIMOM/provider, toute l'information de gestion distribuée sur le domaine géré

5. En fait elle ne diffère que par la formalisation du concept de Provider car celui-ci ne fournit véritablement qu'une fonction d'adaptation telle que définie en d'autres termes dans le Réseau de Gestion des Télécommunications (RGT) en 1992 sous le nom de fonction de médiation et plus tard dans SNMP sous le terme de proxy.

peut être centralisée et mise à la disposition des applications du gestionnaire dans un modèle unique : le modèle CIM.

Cette architecture est certe intéressante, mais dans l'état actuel de la norme, souffre de quelques lacunes et notemment au niveau de l'interface *CIMOM/provider*, qui bien que fondamentale dans l'architecture WBEM, reste encore mal définie. En effet, ni les protocole d'accès, ni les objets maintenus par l'une ou l'autre des entités ne sont spécifiés. Ainsi, la définition de cette interface est actuellement spécifique aux implémentations, ce qui s'oppose à un développement libre des providers en faveur d'un choix forcé sur le CIMOM d'accueil. Par exemple, dans l'implémentation WBEM de Microsoft WMI (Windows Management Instrumentation), le CIMOM ne maintient dans sa base que des objets statiques (i.e. ne varient que très peu), et commuique avec ses providers via le protocole COM/DCOM (Component Object Model / Distributed Component Object Model.).

2.2 Base d'informations de gestion

Le gestionnaire d'objets CIM (CIMOM) maintient une base d'information de gestion (MIB) contenant les données relatives aux ressources du système géré. Cette MIB est composée d'objets gérés CIM (classes et instances) et des dépendances entre ces objets. La description des objets contenus dans la MIB est présentée dans les sections 3 et 5. Le langage de spécification est détaillé dans la section 4. Les opérations sur cette MIB se font via l'interface d'accès CIM présentée dans les sections 6 et 9.

2.2.1 Architecture de la MIB

Dans WBEM l'architecture de la base d'informations de gestion (`CIM Object Repository`) repose sur le concept d'espace de nommage. Un espace de nommage est défini comme un groupement d'objets (classes, instances, relations) dans lequel chaque composant a un nom unique permettant de le distinguer de tous les autres composants de l'espace.

Un espace de nommage peut contenir, outre des objets gérés, un ou plusieurs espaces de nommage. L'arbre issu de cette relation de contenance entre les espaces de nommage définit la hiérarchie de la base d'informations de gestion sur un serveur CIMOM. L'identification d'un espace de nommage au sein d'un serveur est obtenue par la donnée du chemin dans l'arbre de nommage (comme pour l'arborescence des répertoires sous UNIX). Cette architecture est illustrée dans la figure 3.

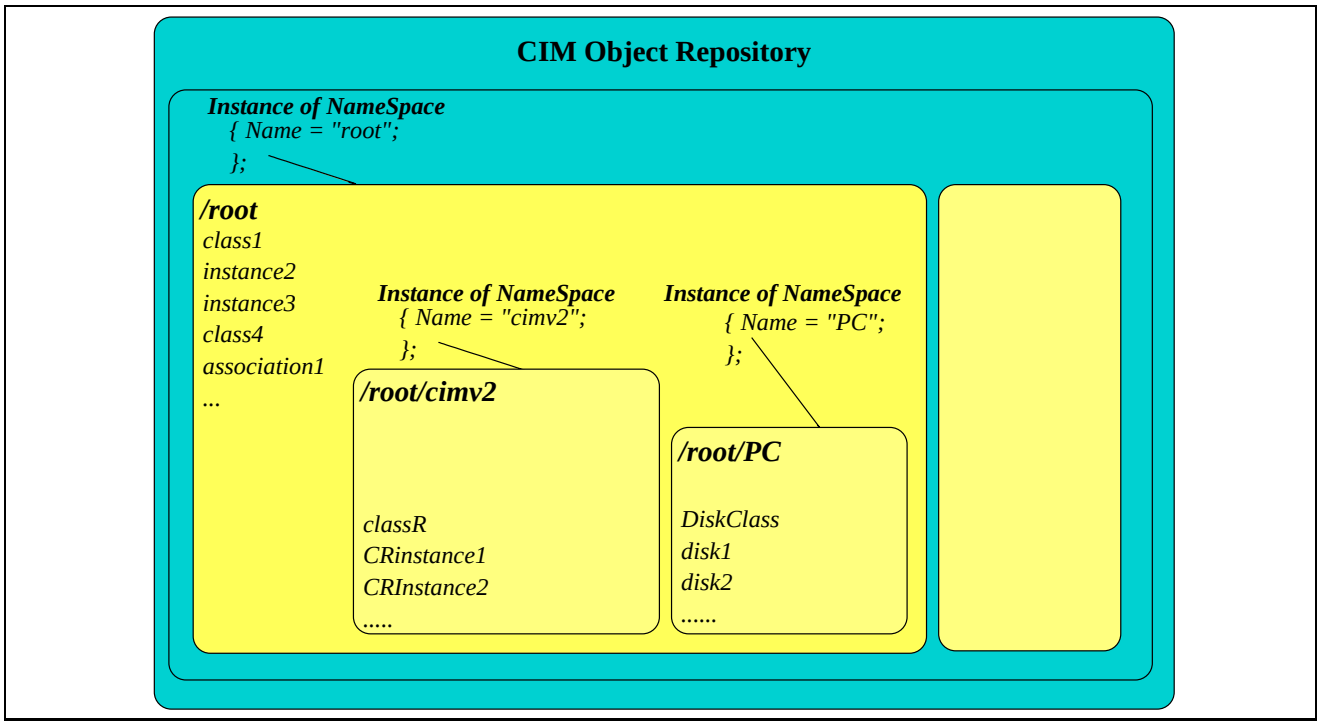


FIG. 3 – *Architecture de la MIB dans un serveur WBEM*

Un espace de nommage peut contenir des instances d'objets gérés mais également des composants de spécification des modèles de l'information.

2.2.2 Nommage des objets

Au sein d'une MIB, toute entité doit être identifiable de manière unique. Aussi existe-il un mécanisme de nommage défini dans ce but au sein de l'approche WBEM.

Le nom d'une classe ou d'un composant de spécification est caractérisé par son label dans la spécification précédé par le chemin dans l'espace de nommage. Un nom d'instance est lui, défini à partir du chemin dans l'espace de nommage suivi du nom de la classe dont l'instance est issue ainsi que des valeurs de certains ou de tous ses attributs qui forment une clef unique d'identification de l'instance (par exemple, un attribut numéro de sécurité sociale formera la clef d'identification d'une personne). La syntaxe exacte des noms est donnée dans la section 3.2.2.

2.3 Résumé

L'architecture WBEM définit quatre rôles génériques pour les entités participant dans une activité de gestion. Ces rôles sont: le **client**, le **serveur**, le **producteur** et le **consommateur**. La généricité des rôles définis permet de dériver de nombreuses configurations à partir de l'architecture et assure une grande flexibilité pour l'intégration de nouvelles composantes notamment en autorisant la hiérarchisation de ces dernières.

À partir de la définition de ces rôles, l'architecture s'affine en définissant le contenu d'un gestionnaire d'objets gérés appelé **CIMOM** ainsi qu'un service de communication permettant aux différentes entités de s'échanger des informations de gestion au travers d'opérations et/ou d'indications.

La base d'informations de gestion maintenue dans une entité CIMOM est organisée de manière hiérarchique grâce au concept d'**espace de nommage**. Ce concept est notamment utilisé pour le nommage des objets gérés au sein d'une MIB.

A première vue, l'architecture WBEM n'apporte pas grand chose de nouveau si ce n'est un renommage, parfois maladroit, des entités et composants intervenants dans la gestion. En effet on y retrouve le concept de gestionnaire, d'agent, de MIB. Cependant, la répartition des rôles est beaucoup plus souple que dans les approches traditionnelles où seuls deux rôles sont définis (gestionnaire et agent).

Dans la suite de ce chapitre nous allons entrer dans les détails des différents composants de l'architecture à savoir le modèle de l'information (langage de spécification + fonctions existantes), le service et le protocole de communication standard.

3 Le modèle de l'information commun CIM

Au coeur de l'initiative WBEM, le modèle de l'information CIM (Common Information Model) est en cours de standardisation par le DMTF[6]. Bien plus qu'une simple collection de ressources modélisées et approuvées par un ensemble de partenaires, CIM a pour objectif de permettre la spécification et la mise à disposition des applications d'une description unifiée et extensible des ressources gérées. Cette modélisation est indépendante des approches existantes. CIM comporte cinq éléments principaux. Ces éléments sont :

1. un méta-modèle décrivant les composants du modèle ;
2. un schéma de nommage des composants gérés ;
3. un langage support pour la spécification des objets gérés ;
4. un modèle de l'information de référence servant de base à la spécification de nouveaux composants gérés ainsi qu'un sous-ensemble représentant les objets que tout serveur doit implémenter ;
5. un ensemble de recommendations sur l'intégration de modèles de l'information issus d'autres approches.

Nous allons dans la suite de cette section décrire brièvement chacun de ces composants. Ils seront par la suite présentés de façon beaucoup plus détaillée.

6. `http://www.dmtf.org`

3.1 Le méta-modèle

Le méta-modèle définit les composants du modèle qui servent de briques de base à la construction des spécifications. CIM adopte une approche orientée-objets pour la spécification des ressources de gestion. Les composants du base du méta-modèle sont :

- le **schéma** qui regroupe un ensemble de spécifications de **classes**, d'**instances** et de **qualifieurs**. Un schéma peut-être vu comme un module SNMP ou une recommandation de l'UIT-T comportant la spécification GDMO d'une fonction de gestion OSI ;
- la **classe** d'objet : toute ressource gérée est modélisée par une ou plusieurs classes d'objets gérés. Une classe est définie par des **attributs** (ou propriétés) et des **méthodes** qui représentent son interface ;
- l'**attribut** représente un état ou une propriété de l'objet dans lequel il est défini. À tout attribut correspond un type de données ainsi qu'une liste d'opérations que l'on peut exécuter sur celui-ci (ex. lire, affecter, ...) ;
- la **méthode** définit une opération que l'on peut invoquer sur une instance d'objet géré. Elle comporte des paramètres formels et une valeur de retour ;
- l'**association** est une classe particulière permettant de spécifier des dépendances entre classes. Une association comprend au moins deux attributs du type **référence** et peut comporter des méthodes et d'autres attributs ;
- la **référence** est un attribut particulier qui permet de pointer vers un objet géré. Celle-ci définit un rôle dans une relation ;
- l'**instance** d'objet est une occurence particulière d'une classe. Il existe deux types d'instances : les instances de classes ou objets gérés et les instances d'associations ou relations (qui sont elles-mêmes des objets gérés) ;
- le **qualifieur** permet de modifier la définition d'un composant (classe, attribut, méthode, instance). Ce composant permet notamment de définir de manière formelle ou semi-formelle des spécificités qui ne rentrent pas dans le méta-modèle sans le modifier. Ceci permet d'affiner de façon simple la sémantique des composants auxquels les qualifieurs sont associés ;
- l'**indication** est une classe particulière dont les instances sont créées par des **déclencheurs** ;
- un **déclencheur** est une opération invoquée lors d'un changement d'état dans une MIB. La définition de son déclenchement est donnée dans un qualifieur pour chaque composant (classe, instance, attribut, méthode) concerné. La spécification précise ainsi que la mise en œuvre de ce dernier n'est pas donnée dans la version actuelle de la norme.

Ces composants forment les éléments de base de la structure de l'information dans l'approche CIM. Les relations entre ces composants sont données dans la figure 4 sous la forme d'un schéma UML.

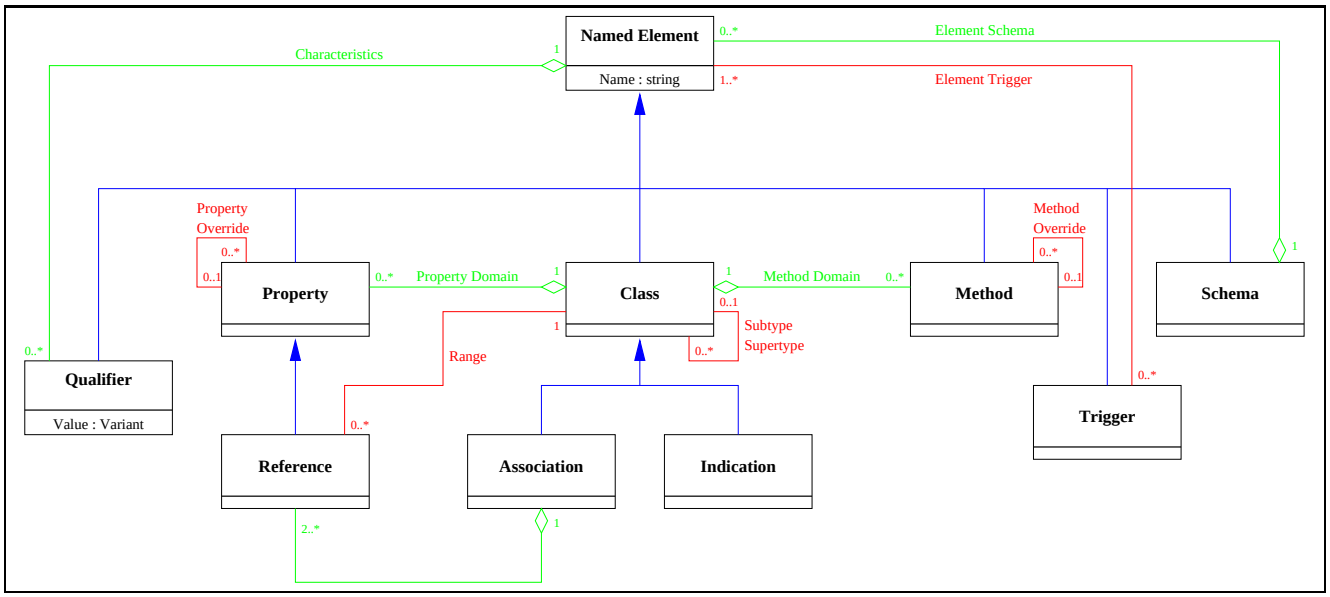


FIG. 4 – *Le méta-modèle CIM*

- Toutes les constructions du méta-modèle héritent de la classe `Meta_NamedElement`[7] qui définit un unique attribut `Name` donnant le nom de l'élément du méta-modèle ;
- L'aggrégation `Meta_ElementSchema` définit le domaine de définition d'un élément du méta-modèle : le schéma. Un schéma peut contenir 0 ou plusieurs éléments. Chaque élément est donné dans le contexte d'un unique schéma ;
- L'aggrégation `Meta_Characteristics` définit le domaine de définition d'une instance de qualifieur : l'élément CIM. A un élément peut être aggrégé 0 ou plusieurs instances de qualifieurs. Chaque instance de qualifieur est aggrégée à un unique élément CIM.
- Une définition d'association est une sous-classe d'une définition de classe CIM ;
- Une définition d'indication est une sous-classe d'une définition de classe CIM ;
- L'association `Meta_SubtypeSupertype` définit la relation d'héritage entre classes. Chaque classe peut hériter des propriétés d'au plus une classe (héritage simple) ;
- Les aggrégations `Meta_PropertyDomain` et `Meta_MethodDomain` définissent le domaine de définition d'une propriété ou d'une méthode : la classe (ainsi que l'association et l'indication par héritage). Chaque classe peut contenir 0 ou plusieurs propriétés (ou méthodes). Chaque propriété (ou méthode) est donnée dans le contexte d'une unique classe ;
- L'association `Meta_PropertyOverride`, resp. `Meta_MethodOverride`, définit la relation de surcharge entre les propriétés, resp. les méthodes. Chaque propriété, resp. méthode, peut surcharger au plus une propriété, resp. méthode;
- Chaque association doit contenir au moins deux références ;

Le méta-modèle CIM est formellement spécifié à l'aide de la notation MOF (Managed Object Format) également utilisée pour la spécification des modèles de l'information. Cette notation est détaillée dans la section 4.

3.2 Nommage des composants

Construire un modèle de l'information dans l'approche CIM consiste à définir un schéma. Ce dernier est composé de classes (objets gérés, associations, indications), d'instances et de qualifieurs. Tout composant au sein d'un schéma doit avoir un nom unique (classes, qualifieurs). L'identification d'une instance se fait via les valeurs des attributs de celle-ci définis comme clefs d'identification.

3.2.1 Orthogonalité schéma / espace de nommage

Dans WBEM, la spécification du modèle de l'information (appelée schéma) et l'espace de nommage sont définis de la même manière comme "ensembles de spécifications" (classe et instances) ce qui peut prêter à confusion. Cette ambiguïté entre le modèle de l'information et la MIB est renforcée car l'on peut définir au sein d'une spécification d'un modèle de l'information, des instances d'objets gérés, qui seront maintenus dans la MIB. De même, la MIB va contenir les spécifications qui sont données dans le modèle de l'information. Afin de lever cette ambiguïté, nous précisons dans cette section les rôles complémentaires de ces deux concepts.

Dans une implémentation de serveur CIMOM, la base d'informations de gestion est organisée de manière hiérarchique suivant le concept d'espace de nommage. Au sein d'un espace de nommage, on trouve des spécifications d'objets gérés ainsi que des instances de ces objets.

La spécification des modèles de l'information qui sont supportés dans une base d'information de gestion repose quand à elle sur la notion de schéma qui regroupe un ensemble de spécifications de classes d'objets gérés, de qualifieurs et d'instances. Un schéma permet de définir un ensemble cohérent de spécifications et de les regrouper sous un nom commun. La définition d'un schéma CIM est généralement apparentée au nom du concepteur ou l'organisme à l'origine du modèle.

Lors de l'implémentation, ces objets sont répartis dans un ou plusieurs espaces de nommage au sein de la MIB ce qui les rend accessibles via les interfaces offertes par le CIMOM et les provider.

Ces deux concepts ne sont donc pas antagonistes ou redondants, mais reposent l'un sur la spécification des objets (schéma), l'autre sur leur localisation dans la hiérarchie de nommage d'une MIB (espace de nommage). La figure 5 illustre la dualité entre le concept de schéma et celui d'espace de nommage.

7. Le préfixe `Meta_` provient des conventions de nommage dans CIM qui requièrent que tout composant soit préfixé par le nom du modèle auquel il appartient suivi de l'*underscore*.

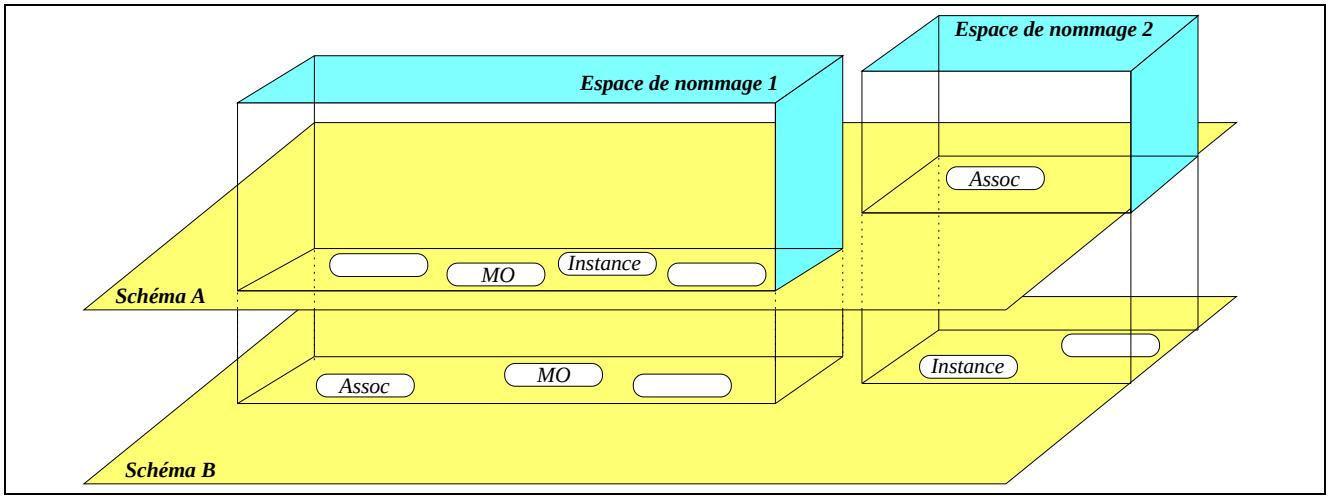


FIG. 5 – *Orthogonalité schéma/ espace de nommage*

3.2.2 L'espace de nommage

Comme nous l'avons vu dans la section 2, un CIMOM maintient les objets dans un ensemble hiérarchique d'espaces de nommages qui ne sont pas liés aux schémas. Tout composant d'une MIB (objet géré, classe, qualifieur) se trouve dans un espace de nommage donné. Comme pour le schéma, tous ces composants doivent avoir un nom unique au sein de l'espace de nommage dans lequels ils se situent. Le nom d'un objet est constitué par l'identificateur de l'espace de nommage auquel il appartient (**Namespace Path**), et par l'identificateur de l'objet dans cet espace de nommage (**Model Path**). La composition d'un nom d'objet est donnée dans la figure 6.

```
1  ObjectName          -> namespace-path " :" model-path
2  namespace-path      -> namespace-type " ://" namespace-handle
3  namespace-type      -> <access-protocol-id>
4  namepsace-handle    -> <cimom-id> ( " /" <ns-label> )+
5  model-path          -> <class-label> ( " ." <property-name> " =" <property-value>
6                                ( " ," <property-name> " = <property-value> )* )
```

FIG. 6 – *Nommage d'un objet CIM*

L'espace de nommage (**NamespacePath**) est identifié par deux paramètres :

- son type (ligne 3) : il identifie le protocole d'accès au CIMOM qui gére l'espace de nommage. Dans l'état actuel de l'art WBEM, les seuls protocoles utilisés sont HTTP et DCOM ;
- son nom global (ligne 4) : il est composé de l'identificateur du cimom (adresse IP) et du nom local dans le CIMOM (liste des noms d'espaces de nommages depuis l'espace de nommage racine).

L'identificateur d'un objet dans un espace de nommage **ModelPath** est défini par deux paramètres :

- Le nom de la classe de l'objet ;
- Une liste de couples (attribut/valeur) qui représentent les valeurs des attributs clefs de l'objet.

La figure 7 illustre un exemple de référence vers une classe et un exemple de référence vers une instance d'objet géré.

Dans le premier exemple, nous donnons la référence vers la classe `CIM_ComputerSystem`, implantée dans l'espace de nommage `/root/cimv2` maintenu par le serveur `dolcourt.loria.fr`. Le protocole d'accès à cette implémentation est le protocole `HTTP`.

Dans le second exemple, nous donnons un exemple de référence vers une instance de la même classe `CIM_ComputerSystem`. Cette instance est identifiée par l'attribut clef `Name` avec la valeur "`dolcourt.loria.fr`".

Exemple de référence vers une classe

```
1 HTTP ://dolcourt.loria.fr /root /cimv2 :CIM_ComputerSystem
```

Exemple de référence vers une instance

```
2 HTTP ://dolcourt.loria.fr /root /cimv2 :CIM_ComputerSystem .Name=''dolcourt.loria.fr''
```

FIG. 7 – *Exemples de nommages d'objets CIM*

3.3 Le langage de spécification

Afin de permettre la construction de nouvelles spécifications et de fournir un support textuel pour la formalisation des modèles de l'information, le langage MOF (Managed Object Format) a été développé dans l'approche CIM.

Ce langage fournit une syntaxe et une sémantique en lien avec le méta-modèle CIM permettant de spécifier tout composant d'un modèle de l'information (qualifieur, classe, association, ...). Sa syntaxe est définie dans la section 4.

3.4 Le modèle commun

Disposer d'un formalisme commun pour la spécification des ressources gérées constitue un premier pas vers l'intégration et l'interopérabilité en gestion de réseaux. Cette condition nécessaire n'est cependant pas suffisante. Il faut en effet avoir également une modélisation uniforme des mêmes ressources. Ceci est assuré par la définition d'un modèle de l'information commun. Ce modèle existe dans WBEM. Il est spécifié dans CIM et définit les composants de base de tout modèle de l'information. Le standard publié par le DMTF définit deux modèles de références correspondant chacun à un niveau d'abstraction dans la gestion :

- le modèle *Core* définissant les classes et associations de base génériques à tout domaine de gestion ;
- le modèle *Common* définissant les classes et associations utilisables dans différents domaines de la gestion indépendamment de toute implémentation. Le modèle *Common* se décompose en 5 sous-modèles : `System`, `Application`, `Network`, `Device` et `Physical`.

Ces modèles sont détaillées dans la section 5.

3.5 Les propositions de techniques d'intégration

La gestion des réseaux ne naît pas avec l'approche WBEM ni avec CIM. Aussi, dans le cadre du modèle commun CIM, différentes techniques de mise en correspondance et de traduction des modèles de l'information issus de différentes approches standardisées ou normalisées sont proposées pour être implémentées dans les providers, et ce afin de présenter au CIMOM l'information de gestion dans le modèle CIM.

Ces techniques sont au nombre de trois. La première propose une expression du méta-modèle de la technique à intégrer dans le méta-modèle CIM. La seconde propose une mise en correspondance des éléments du méta-modèle source avec ceux du méta-modèle CIM. La dernière propose un mapping des spécifications issues de la technique source vers les spécifications du schéma commun CIM. Ces trois techniques font l'objet d'une section spécifique dans ce chapitre. Elles sont détaillées dans la section 8.

4 Le langage de spécification des ressources gérées

Afin de permettre à tout concepteur de système de gestion de spécifier ses propres objets gérés, l'approche WBEM propose un langage de spécification issu des travaux du DMTF. Le langage s'appelle Managed Object Format (MOF) et est intégré au modèle CIM [8]. Il permet de décrire sous une forme textuelle, les différents composants d'un modèle de l'information ou schéma.

Les composants de base d'une spécification sont : la classe, le qualifieur, l'instance et l'association. À chaque composant est associée une syntaxe particulière dans le langage.

La syntaxe associée à ces différents composants est détaillée dans cette section. La description de chaque composant est illustrée par des exemples d'utilisation.

4.1 La spécification de classe

La classe est le composant de base d'un modèle de l'information. Une classe est définie par un ensemble (optionnel) de qualifieurs, par son nom, la classe dont elle hérite les propriétés, un ensemble d'attributs (propriétés dans la terminologie WBEM), et de méthodes.

Les règles d'héritage sont les suivantes :

- pour les propriétés :

  on peut définir dans une sous-classe de nouveaux attributs. Il est également possible de redéfinir des attributs hérités d'une super-classe. Cette redéfinition ne peut se faire qu'en substituant le type de l'attribut hérité par un type compatible dans la sous-classe. La compatibilité des types exige que le type associé à l'attribut dans la sous-classe soit un sous-type du type associé à l'attribut dans la super-classe, par exemple, une chaîne de caractères alphabétiques est un sous-type d'une chaîne de caratères alphanumériques. Ceci garantit la covariance de type ;
- pour les méthodes :

  On peut définir dans une sous-classe de nouvelles méthodes. Il est également possible de redéfinir des méthodes héritées dans une sous-classe. Cette redéfinition ne peut se faire qu'en conservant la covariance des résultats et la contravariance des arguments, i.e. le type du résultat de la méthode dans la sous-classe doit être un sous-type de celui défini dans la super-classe et les types des paramètres de la méthode dans la sous-classe doivent être des super-types de ceux définis pour la méthode dans la super-classe.

De plus, toute redéfinition doit être explicitement spécifiée pour tout composant (classe, attribut, méthode) via un qualifieur spécifique appelé `Override` (voir section 4.2).

4.1.1 Les éléments du langage

Les règles de production associées à la syntaxe de définition de classe sont données dans la figure 8.

Une définition de classe tout comme ses composants peut être affinée par un ou plusieurs qualifieurs (voir section 4.2). Ceux-ci sont donnés dans le préfixe de la définition de classe. Au sein de ce préfixe, les qualifieurs sont séparés entre eux par une virgule.

```
1  [ qualifiers ]  CLASS <schema-name>_<class-label>
2         [ AS <alias-identifier> ] [ : <super-class-label> ]
3            {
4            [  property |  method ]*
5            }
6  ;
7
8  property -> [ qualifiers ]
9                 dataType <property-label> [  array ] [ default-value ]  ;
10 method -> [ qualifiers ]
11               dataType <method-label>  "("  parameter [  ","  parameter ]*  ")" ";"
12 parameter -> [ qualifiers ] ( dataType | objectRef ) <parameter-label> [ array ]
13
14 array ->  "[" [ <positiveDecimalValue> ]  "]"
15 default-value ->  =  value |  arrayValue
16
17 value -> simpleValue
18         | objectPath
19 arrayValue ->  "{"  value [ ","  value ]*  "}"
20 qualifiers ->  "["  qualifier [ ","  qualifier ]*  "]"
```

FIG. 8 – *La spécification de classe*

À chaque classe correspond un identificateur unique composé par le nom de la classe et par le nom du schéma auquel elle appartient : <schema-name>_<class-label> (ligne 1). En plus du nom usuel, on peut associer un nom d'alias à une classe : clause "AS <alias-identifier>" (ligne 2). Celui-ci définit un second nom qui peut être utilisé dans la même spécification MOF pour référencer la classe lors de la compilation.

Une classe peut hériter des propriétés et des méthodes d'une super-classe. Cela se spécifie dans la clause "`: <superclass-label>`" (ligne 2) dans laquelle on donne le nom de la classe dont on hérite les propriétés. Cette clause est optionnelle. Les règles d'héritage ont été présentées au début de la section.

Chaque classe comporte éventuellement des attributs (*property* dans la terminologie WBEM). La syntaxe de leur définition est donnée dans les lignes 8 et 9. Une propriété est définie par un ensemble (optionnel) de qualifieurs, un type de base, un nom, sa nature (propriété simple ou extension du type à un tableau) ainsi que par une valeur par défaut optionnelle (clause *default-value*). La valeur par défaut est affectée à la propriété lors d'une instanciation sans affectation directe. La portée d'une définition de propriété est celle de la classe dans laquelle elle est définie, ainsi que chacune de ses sous-classes.

Une classe peut contenir des méthodes. La syntaxe associée à la définition de méthodes est donnée dans les lignes 10 et 11. Une méthode est définie par un ensemble (optionnel) de qualifieurs et par sa signature, i.e. le type de donnée qu'elle retourne, son nom ainsi que l'ensemble de ses paramètres. Chaque paramètre est défini par un ensemble (optionnel) de qualifieurs, par son nom et par son type.

4.1.2 Un exemple de spécification

La figure 9 comporte des exemples de spécifications de classes. Ces spécifications sont issues du modèle de base du DMTF (`Core Schema`) et décrit trois classes liées à la définition des composants logiques d'un système[8].

```
1  [Abstract ]  CLASS CIM_LogicalElement
2    {
3    };
4
5  [Abstract ] CLASS CIM_System : CIM_LogicalElement
6    {
7     [Key] string Name;
8     string NameFormat;
9     uint32 LocalId;
10   };
11
12   [Description("Computer system logical model")  ]
13 CLASS CIM_ComputerSystem : CIM_System
14   {
15    [Override("LocalId")] uint32 LocalId;
16    boolean Reboot(datetime Time);
17   };
```

FIG. 9 – *Un exemple de spécification de classes*

La première classe `CIM_LogicalElement` (lignes 1 à 3) est une classe abstraite qui ne peut être instanciée (présence du qualifieur `Abstract`, voir section 4.2). Cette classe appartient au schéma de base CIM. Elle représente un composant logique et ne comporte aucune propriété ni méthode.

La seconde classe `CIM_System` (lignes 5 à 9) est elle aussi abstraite et modélise un composant logique : le système. Cette classe définit trois attributs, deux de type chaîne de caractères et un de type entier. Le premier comporte le nom du système, et servira de clef pour le nommage des instances de la classe (présence du qualifieur `Key`, voir section 4.2). Le second comporte le format utilisé pour donner le nom de l'instance (adresse IP par exemple). Le dernier attribut est un entier et représente un identificateur local de l'équipement pour les besoins de l'adminstrateur.

La dernière classe `CIM_ComputerSystem` (lignes 11 à 16) modélise la partie logique d'un système informatique. Cette classe hérite des propriétés d'un système. Elle ne comporte qu'un unique attribut de type entier. Le qualifieur `Override` donné à l'attribut, indique que la définition surcharge la définition donnée dans la super-classe. Elle comporte aussi une unique méthode `Reboot` qui prend en paramètre une date, et renvoie un booléen indiquant si l'opération a été prise en compte ou pas.

8. Les définitions données dans l'exemple ont été modifiées par rapport à celles données par le DMTF pour illustrer les différents éléments du langage MOF

4.2 Le qualifieur

Un qualifieur est un composant permettant de définir une propriété sur un élément. Il est généralement utilisé pour associer une sémantique à une définition de classe, d'instance, de méthode, d'attribut ou de n'importe quel composant d'une spécification MOF, excepté le qualifieur lui-même. Le langage MOF permet la définition de types de qualifieurs qui seront par la suite utilisés pour qualifier un composant d'une spécification.

4.2.1 La syntaxe

Les règles de production pour la définition d'un qualifieur sont données dans la figure 10 ci-dessous.

```
1   QUALIFIER <identifier>
2               : DataType [  array ]
3               [  =   defaultValue ]
4               "," SCOPE "("  metaElement [ ","  metaElement ]*  ")"
5               [ "," FLAVOR "("  flavor [ ","  flavor ]*  ")" ]
6             ";"
7
8  array ->  "[" [ <positiveDecimalValue> ]  "]"
9
10 defaultValue ->  simpleInitializer |  arrayInitializer
11
12 metaElement -> SCHEMA | CLASS | ASSOCIATION
13           | INDICATION | METHOD | PROPERTY
14           | REFERENCE | ANY
15
16 flavor -> ENABLE_OVERRIDE | DISABLE_OVERRIDE
17          | RESTRICTED | TO_SUBCLASS | TRANSLATABLE
```

FIG. 10 – *La spécification de qualifieur*

Un qualifieur est défini par un nom, un type de donnée qui peut être un tableau (clause *array*, ligne 2). On peut lui associer une valeur par défaut : clause *defaultValue* (ligne 3). Pour chaque qualifieur, on définit également son champ d'application, à savoir les composants d'une spécification auxquels il s'applique (clause **SCOPE**). Les composants cibles peuvent être un schéma, une classe, une association, une indication, une propriété, une méthode, un paramètre, une référence, ou une instance.

La dernière composante de la spécification d'un qualifieur concerne sa validité vis-à-vis de la propagation, i.e. l'héritage. Le spécifieur d'un qualifieur peut ici définir les règles de propagation de celui-ci en précisant si le qualifieur peut être instancié différemment (**ENABLE_OVERRIDE**) ou non (**DISABLE_OVERRIDE**) dans une sous-classe. Il peut également préciser la validité du qualifieur en déclarant si celui-ci n'est valable que dans la classe dans laquelle il est utilisé (**RESTRICTED**) ou s'il est automatiquement propagé aux sous-classes de la classe déclarante (**TO_SUBCLASS**).

4.2.2 Un exemple de définition

Illustrons la notion de qualifieur au travers de deux exemples. Nous désirons attacher à nos spécifications des informations supplémentaires qui sont un comportement informel sous forme de chaîne de caractères ainsi qu'une référence vers le concepteur d'un schéma. Pour cela, on spécifie deux qualifieurs: `Behaviour` et `CreatorInfo`. Les spécifications sont données dans la figure 11.

```
1  QUALIFIER Behaviour : String  =  null  ,
2          SCOPE (CLASS, ASSOCIATION, INDICATION, METHOD, PROPERTY, REFERENCE)  ,
3          FLAVOR (TO_SUBCLASS)  ;
5
6  QUALIFIER CreatorInfo  : String [3]  =  {null,null,null}  ,
7          SCOPE (CLASS, ASSOCIATION, INDICATION)  ;
```

FIG. 11 – *Exemples de définitions de qualifieurs*

Le qualifieur de comportement est défini par une chaîne de caractères dont la valeur par défaut est `null`. Ce qualifieur s'applique aux spécifications de classes, d'association, d'indication, de méthode, d'attribut et de référence. Il se propage naturellement à l'ensemble des sous-classes de la classe dans laquelle il est utilisé.

Le second qualifieur permet d'associer des informations sur le créateur d'un élément d'un schéma CIM. Ce qualifieur est défini comme un tableau de 3 chaînes de caractères dont les valeurs par défaut sont `null`. Ce qualifieur ne s'applique qu'aux éléments de type classe, association ou indication, il ne peut donc pas s'appliquer aux attributs et méthodes.

4.2.3 Un exemple d'utilisation

Une fois défini, un qualifieur peut être instancié pour qualifier un composant d'une spécification. Des exemples d'instances de qualifieurs sont donnés dans la spécification ci-dessous (figure 12).

```
1   [ CreatorInfo("Olivier Festor","INRIA Lorraine","festor@loria.fr")]
2 INRIA_ClasseExemple CLASS
3 {
4     [ Behaviour("cet attribut est toujours à 0")]
5   uint16 proprieteExemple = 0;
6 };
```

FIG. 12 – *Exemple d'instanciation de qualifieurs*

Si un qualifieur n'est pas explicitement instancié (incluant l'instanciation par héritage) dans une spécification appartenant à son champs d'application (`Scope`), il sera implicitement attribué à la spécification en prenant sa valeur par défaut si elle existe.

Si un qualifieur est attribué à une spécification sans aucune valeur (explicite ou par défaut), il prendra la valeur `TRUE` pour les qualifieurs de type booléen, le vecteur vide pour ceux de type vecteur, et `NULL` pour ceux de tout autre type.

4.2.4 Les qualifieurs standards de l'approche CIM

CIM définit un certain nombre de qualifieurs de base. Ceux-ci sont regroupés en deux catégories : les qualifieurs du méta-modèle et les qualifieurs standards. Les qualifieurs du méta-modèle permettent d'affiner les définitions des méta-éléments. Ils sont au nombre de deux. Le premier, `Association`, est un booléen qui, associé à une classe, précise qu'elle définit une association. Le second, `Indication`, permet d'affiner une définition de classe, indiquant que celle-ci spécifie une indication.

Les qualifieurs standards sont plus nombreux que les qualifieurs de méta-modèle (le DMTF en définit environ 50). Les plus importants sont :

- `Abstract(boolean)` : associé à une classe, il permet de spécifier que celle-ci ne peut être instanciée (voir l'exemple de la figure 9) ;
- `Key(boolean)` permet de préciser qu'un attribut (propriété d'une classe d'objets ou référence d'une association) est utilisé comme clef pour le nommage des instances ;
- `Description(string)` permet d'associer une description textuelle à un composant d'une spécification ;
- `Override(string)` permet de préciser que l'élement surcharge un autre élément défini dans une superclasse, et donne l'origine de la définition surchargée ;
- `Read(boolean), Write(boolean)` permettent de spécifier si un attribut ou une référence peuvent être lus, respectivement affectés.

La liste complète des qualifieurs standards définis dans l'approche est donnée dans [8].

4.3 L'association

Une association est définie comme une classe d'objet particulière. Elle permet de relier plusieurs objets gérés et de ce fait comporte en plus des composants génériques d'une classe (propriétés et méthodes), des attributs particuliers qui maintiennent des références vers les objets gérés que l'association relie. Une association ne peut pas hériter d'une classe qui n'est pas une association, et toute sous-classe d'une association est forcémment une association.

Les règles d'héritage pour les références sont les suivantes :

- On ne peut pas changer l'arité d'une super-association. L'arité étant définie comme le nombre de références spécifiées dans une association.
- La redéfinition d'une référence (surcharge) est possible par utilisation du qualifieur `Override`. La nouvelle définition doit référencer une sous-classe de la classe référencée dans la super-association.

4.3.1 Syntaxe

Une déclaration d'association comporte une syntaxe proche de celle d'une classe. La distinction première se fait au niveau des qualifieurs. En effet, toute super-association (racine dans l'arbre d'héritage) comporte dans ses qualifieurs, le méta-qualifieur spécifique `Association)` qui indique que la classe définit une association. Le qualifieur `Association` n'est obligatoire que pour la super-association, toute sous-classe de celle-ci sera considérée comme association. La seconde différence concerne les composants de la classe. En effet une association doit posséder (ou hériter) au moins deux attributs références vers des objets gérés. La syntaxe de declaration est définie dans la figure 13.

```
1  "[" [ Association ],  qualifiers  "]"  CLASS <class-label>
2         [ AS <alias-identifier> ] [ : <super-class-label> ]
3            {
4            [ property |  method |  reference ]*
5            }
6  ;
7
8   reference -> [ qualifiers ]  objectRef <reference-name>
9                                      [ default-value ]  ;
10   objectRef -> <class-label>   "REF"
```

FIG. 13 – *Syntaxe de déclaration d'association*

Une référence est définie par un ensemble optionnel de qualifieurs, un label <`reference-name`> (ligne 8) qui associe un nom au rôle, ainsi que par le nom de la classe <`class-label`> dont les instances peuvent être référencées par le rôle.

Dans une instance d'association, chaque référence comportera un identificateur d'instance d'objet décrivant l'objet référencé par le rôle.

Parmi les qualifieurs standards définis par le DMTF, certains sont spécifiques aux associations et aux références, et permettent de préciser la sémantique de la relation :

- Le qualifieur `Aggregation(boolean)` associé à une spécification de relation, indique que celle-ci est de type aggregation (voir sec 4.3.5) ;
- Le qualifieur `Aggregate(boolean)` associé à une référence permet de spécifier le rôle d'aggrégat dans une aggrégation (voir sec. 4.3.5) ;
- Les qualifieurs `Min(int)` et `Max(int)` associés à une spécification de référence, permettent de définir la cardinalité du rôle dans une association. La cardinalité d'un rôle étant définie par le nombre d'instances d'objets gérés qui peuvent tenir le rôle pour l'association ;
- Les qualifieur `Weak(boolean)` associé à une référence permet de spécifier le rôle faible dans une association faible (voir sec. 4.3.3).

4.3.2 Un exemple d'association

Nous présentons ici un exemple simple de déclaration d'association extrait du schéma `Core` défini par le DMTF. La spécification est donnée dans la figure 14.

Dans cet exemple, l'association définit une relation de dépendance entre deux objets gérés. Pour cela, l'association définit deux rôles : un rôle "objet supérieur" (**Antecedant**, ligne 3) et un rôle "objet dépendant" (**Dependant**, ligne 4) qui va référencer un objet dépendant de celui référencé dans le rôle `Antecedant`. Cette association est abstraite (présence du qualifieur `Abstract`), et ne peut pas être instanciée directement. Elle permet par contre de donner un cadre générique pour la définition de relations de dépendances entre objets gérés. Une relation de dépendance concrète est donnée dans la section qui suit (l'association CIM_HostedService, figure 15).

```
1   [ Association, Abstract]
2  CLASS CIM_Dependency
3  {
4     [Description("Reprensents the independant object")]
5   CIM_ManagedSystemElement  REF Antecedent  ;
6     [Description("Represents the object dependent on the Antecedent object")]
4   CIM_ManagedSystemElement  REF Dependent  ;
5  };
```

FIG. 14 – *Un exemple de spécification d'association*

4.3.3 Les associations faibles

Les associations dites faibles sont définies dans CIM pour représenter une relation de contenance comparable à celle définie pour le nommage dans le modèle de gestion OSI (`Name-Binding`). Pour cela, elle identifie un (ou plusieurs) rôle(s) "fort(s)" dans l'association et un et un seul rôle "faible". Les objets gérés "faibles" seront alors contenus dans des objets gérés "forts", i.e ne pourront exister dans la MIB que dans le contexte des objets gérés "forts".

Une et une seule classe peut être spécifiée comme "faible" dans une association faible, à l'aide du qualifieur `Weak` attaché à la référence vers cette classe. Les attributs clefs des autres classes référencées dans l'association (dites classes "fortes" de l'association) devront alors être propagés vers la classe "faible", i.e inclus dans la liste des attributs clefs de la classe faible. À chacun de ces attributs propagés, sera associé le qualifieur `Propagated` permettant de désigner la classe forte à l'origine de la définition de l'attribut. Ainsi, une instance de la classe faible sera nommée dans le contexte des instances des classes fortes. Par exemple, un disque dur sera nommé dans le contexte d'une machine hôte.

4.3.4 Un exemple d'association faible

Pour illustrer la spécification d'une association faible, nous présentons ici un exemple tiré du modèle de base défini par le DMTF. Cet exemple comprend la déclaration d'une association binaire faible ainsi que la spécification de chacune des classes participantes à l'association. Les classes et l'association données dans la figure 15 ont été allégées par rapport aux spécifications du DMTF.

```
1 class CIM_Service : CIM_LogicalElement
2 {
3     [ Key]
4   string Name ;
5     [ Propagated("CIM_System.Name"), Key]
6   string SystemName ;
7 };

8 [ Association]
9 class CIM_HostedService : CIM_Dependency
10 {
11     [Override("Antecedent")]
12   CIM_System REF Antecedent ;
13     [Override("Dependent"), Weak]
14   CIM_Service REF Dependent ;
15 };
```

FIG. 15 – *Un exemple de spécification d'association faible*

L'association **CIM_HostedService** (lignes 8 à 15) permet de relier une modélisation de service (instance de **CIM_Service**, lignes 1 à 7) à une modélisation du système dans lequel il est implanté (instance de la classe **CIM_System** spécifiée dans la figure 9). Dans cette association, le rôle faible est attribué à la classe CIM_Service à l'aide du qualifieur `Weak` (ligne 13) signifiant que tout service doit être nommé dans le contexte du système qui l'implante. Pour cela, on retrouve dans la spécification de CIM_Service un attribut clef propagé (attribut SystemName, lignes 5 et 6). La spécification de cette propagation d'attribut clef est donnée par le qualifieur `Propagated`.

Cette notion d'association faible se traduit au niveau de la base de gestion par des relations de contenance entre instances d'objets gérés. Ceci est illustré dans la figure 16. Dans ce schéma, deux instances de CIM_Service (`serviceA` et `serviceB`) sont créées et identifiées dans le contexte d'une instance de CIM_ComputerSystem (sous-classe de CIM_System).

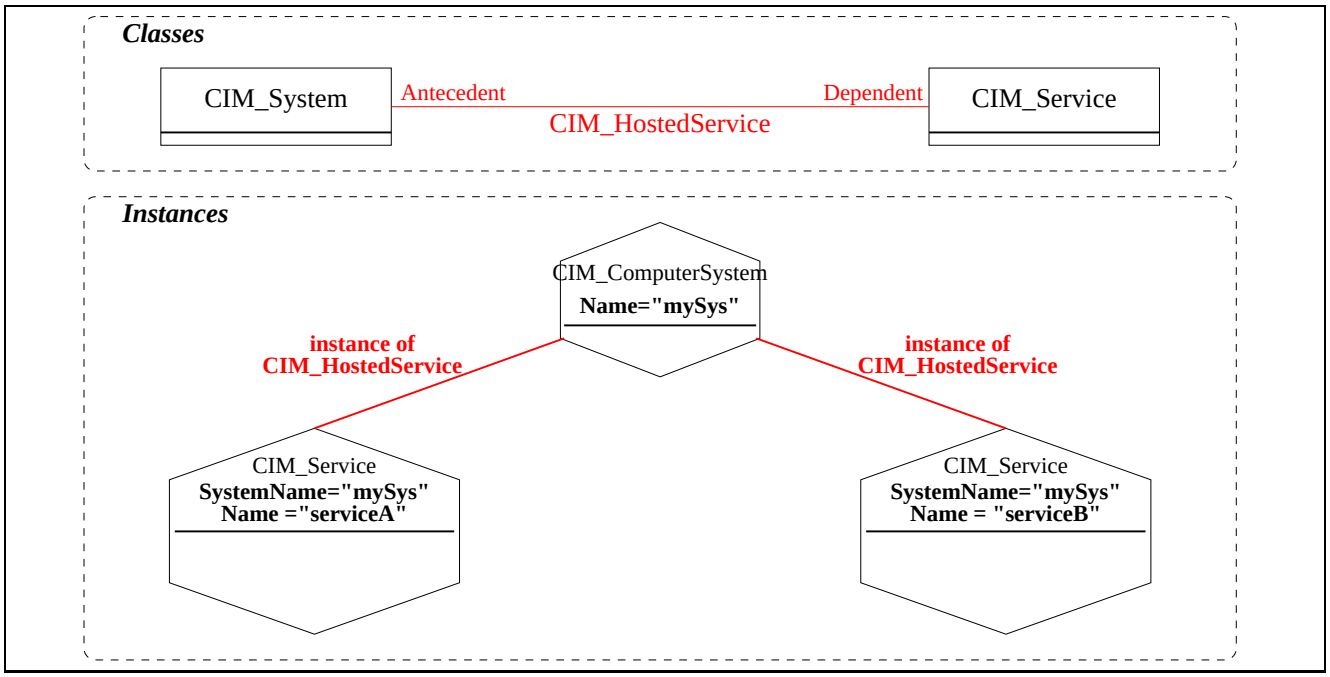


FIG. 16 – *Un exemple d'association faible*

4.3.5 Les aggrégations

Une aggrégation est définie comme un type particulier d'association : elle permet de définir une relation de composition entre objets. L'aggrégation permet alors de spécifier une entité (appelée aggrégat) comme la composition d'une (ou plusieurs) autre(s) entité(s) (appelée(s) entité(s) aggrégée(s)). Par exemple, dans la spécification du méta-modèle CIM, la classe est définie comme aggrégation (i.e. composition) d'une ou plusieurs propriété(s) et d'une ou plusieurs méthode(s).

Le modèle CIM offre la possibilité de spécifier ce type d'associations en s'appuyant sur deux qualifieurs spécifiques : `Aggregation` et `Aggregate`. Le premier permet de donner la sémantique d'aggrégation à une définition d'association, et le second permet de spécifier le rôle d'aggrégat dans cette association. Un exemple de spécification MOF d'une aggrégation, tirée du schéma de base CIM, est donné dans la figure 17.

```
1  [ Association,  Aggregation,  Abstract]
2 class CIM_Component
3    {
4       [ Aggregate, Description("The parent element")]
5     CIM_ManagedSystemElement REF GroupComponent ;
6       [Description("The child element")]
7     CIM_ManagedSystemElement REF PartComponent ;
8    };
```

FIG. 17 – *Un exemple de définition d'aggrégation*

4.4 L'instance

Une instance CIM est caractérisée par la classe (d'objet géré ou de relation) dont elle est issue, l'ensemble des attributs (propriétés et/ou références) qu'elle comporte ainsi que les valeurs qui leurs sont associées. Tout comme pour les classes, on peut associer un ensemble de qualifieurs à une instance et à ses attributs.

4.4.1 La syntaxe

La spécification d'une instance est définie par un ensemble de qualifieurs, par la classe qu'elle instancie , par la (les) valeur(s) prises par le(s) attribut(s) clef(s) (propriétés ou références) de la classe et par la liste (optionnelle) des valeurs prises par ses attributs simples (autres que les attributs clefs). La syntaxe de déclaration d'une instance est donnée dans la figure 18.

```
[ qualifiers] INSTANCE OF <class-label>
    [ AS <alias-identifier> ]
{
[ [ qualifiers] ( <property-label> | <reference-label> ) = initializer ; ]+
};

initializer -> simpleInitializer | arrayInitializer
               | aliasIdentifier | ObjectPath

aliasIdentifier -> $<label>
```

FIG. 18 – *Syntaxe de déclaration d'instance*

Chaque attribut ou référence clef (spécifiés avec le qualifieur `Key`) de la classe doit être initialisé dans la déclaration de l'instance. L'initialisation des autres attributs ou références de la classe est optionnelle lors de la déclaration de l'instance. Si un attribut n'a pas été affecté dans la déclaration, celui-ci prendra la valeur par défaut éventuellement définie dans la classe, sinon la valeur `NULL`.

4.4.2 Un exemple

Prenons l'exemple de l'initialisation d'un objet de type `CIM_ComputerSystem` dont la définition de classe est donnée dans la figure 9 (section 4.1). Dans cet exemple, seulement deux attributs sont initialisés. Les autres attributs de la classe, ne possédant pas de valeur d'initialisation par défaut, sont initialisés à `NULL`.

```
INSTANCE OF CIM_ComputerSystem
{
      Name = "152.81.11.87";
      NameFormat = "IP";
      LocalId = 12;
} ;
```

FIG. 19 – *Un exemple de définition d'instance*

Prenons maintenant un exemple de définition d'instance d'association. Celui-ci illustre une instanciation de l'association `CIM_HostedService`. La spécification de l'association est donnée dans la figure 15. Dans cet exemple, les deux rôles sont initialisés à des identificateurs d'instances des classes référencées dans la définition.

4.5 Les directives de compilation

Comme nous l'avons évoqué au début de la présentation du langage, il existe des directives de compilation particulières. Ces directives sont spécifiées par une clause intitulée `#pragma`. Elles sont utilisées pour donner à un compilateur ou un chargeur de spécifications MOF inclu dans un serveur, des informations sur la nature

```
1  INSTANCE OF CIM_HostedService
2    {
3     Antecedant="/root/example:CIM_ComputerSystem.Name=\"152.81.11.87\""
4     Dependant="/root/example:CIM_Service.Name=\"myService\",SystemName=\"152.81.11.87\""
5    } ;
6
7  INSTANCE OF CIM_HostedService
8    {
9     Antecedant="/root/example:CIM_ComputerSystem.Name=\"152.81.11.87\""
10     Dependent="/root/example:CIM_Service.Name=\"serviceB",SystemName=\"152.81.11.87\""
11   } ;
```

FIG. 20 – *Un exemple de définition d'instance d'association*

(généralement la localisation) des spécifications qui succèdent la directive dans le fichier MOF. Les primitives de compilation standards définies par le DMTF sont au nombre de 8 :

1. **#pragma include(**<file-name>**)** : indique au compilateur d'inclure les spécifications du fichier MOF désigné par <file-name> au niveau de la déclaration de la directive dans le fichier en cours de traitement ;
2. **#pragma locale(**<locale>**)** : indique au compilateur la langue utilisée pour les déclarations (chaînes de caractères) du fichier. En l'absence de cette directive, la langue prise par défaut est "en_US" (anglais Américain) ;
3. **#pragma instancelocale(**<locale>**)** : indique au compilateur que pour les déclarations d'instances, les valeurs prises par les propriétés de type chaîne de caractères sont faites dans la langue spécifiée en paramètre de la directive ;
4. **#pragma namespace(**<namespace-path>**)** : indique au compilateur l'espace de nommage dans lequel doivent être implantés les définitions (classes et instances) qui succédent la directive ;
5. **#pragma nonlocal(**<namespace-path>**)** : indique au compilateur l'identification (Namespace Path) de l'implémentation effective des instances référencées dans le fichier, dans le cas où celles-ci ne sont pas définies dans le fichier (voir sec 4.5.2) ;
6. **#pragma nonlocaltype(**<namespace-type>**)** : indique au compilateur le protocole d'accès au CIMOM (Namespace Type) pour obtenir l'accès à l'implémentation effective des instances référencées dans le fichier ;
7. **#pragma source(**<namespace-path>**)** : indique au compilateur la localisation de l'implémentation de CIM à l'origine des définitions qui succèdent la directive (voir sec. 4.5.2) ;
8. **#pragma sourcetype(**<namespace-type>**)** : indique au compilateur le type de l'implémentation de CIM à l'origine des définitions qui succèdent la directive ;

4.5.1 Les mécanismes *import/export*

Les directives citées ci-dessus, sont fondamentalement utilisées pour les mécanismes *import/export* définis dans WBEM pour la migration "massive" de l'information de gestion d'une plate-forme à une autre. En effet, dans son souci d'instrumenter une approche de gestion indépendante de toute technologie d'implémentation, le DMTF a proposé ces deux mécanismes afin de permettre le partage de l'information entre les plates-formes et par conséquent multiplier les points d'accès aux objets gérés indépendemment de leur implémentation.

Le double mécanisme *import/export* permet de charger la base de gestion d'une plate-forme par un sous-ensemble (ou tout l'ensemble) de l'information maintenue dans la base de gestion d'une autre plate-forme. La figure 21 illustre l'utilisation de ces mécanismes entre deux plates-formes.

Le mécanisme *export* permet de générer un fichier MOF représentant un sous-ensemble de l'information maintenue dans la base de gestion d'un serveur (plate-forme A). Le mécanisme *import* permet de charger l'information de gestion dans un serveur (plate-forme B) depuis un fichier MOF. Ces deux mécanismes sont complétés par des opérations de mise à jour effectuées par la seconde plate-forme via l'interface de gestion offerte par la première plate-forme.

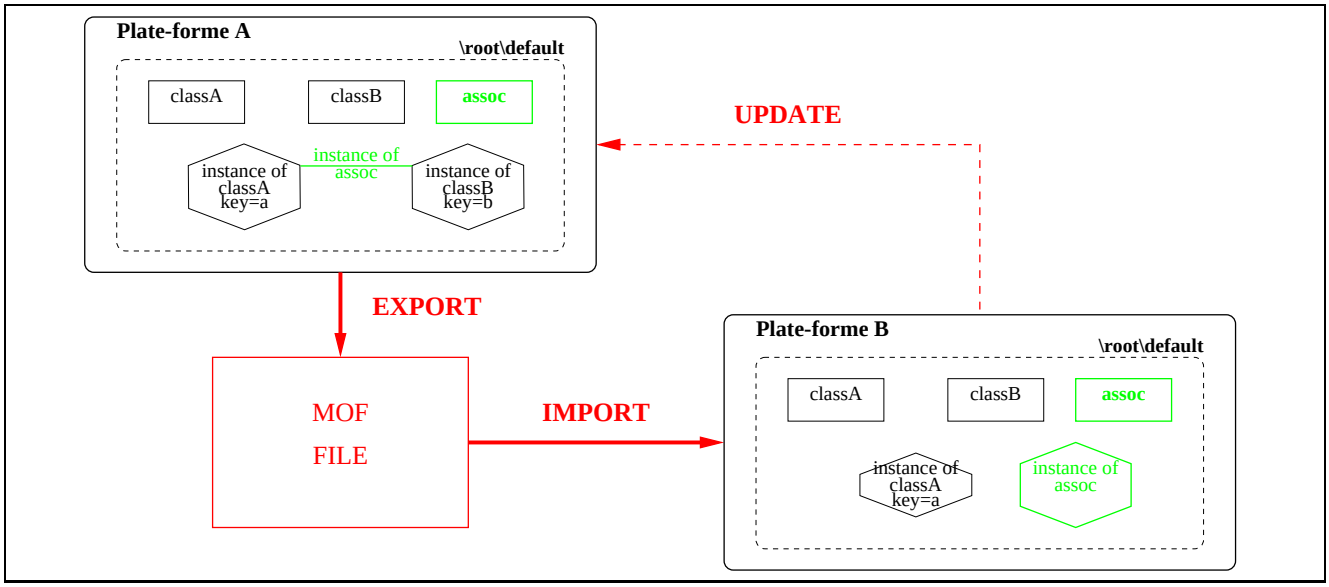


FIG. 21 – *La migration de l'information de gestion*

4.5.2 Illustration de l'utilisation des clauses `#pragma`

Lors de ces opérations *import/export*, les fichiers MOF échangés sont complétés par des informations sur la localisation des objets permettant à la plate-forme destination d'accèder aux objets maintenus dans la base de gestion de la plate-forme source lors de la phase de mise à jour. Une illustration de ces directives dans un fichier MOF est donnée dans la figure 22.

```
1  #pragma namespace( "/root/default")

2  class  classA {
3    [Key] string  key ;
4  } ;
5  class  classB {
6    [Key] string  key ;
7  } ;
8  [Association] class  assoc {
9     classA REF  roleA ;
10     classB REF  roleB ;
11 } ;

12  #pragma source( "HTTP://serveurA/root/default")
13 instance of  classA {
14    key = "a" ;
15 } ;
16 instance of  assoc {
17    roleA = "classA.key=a" ;
18      [ NonLocal( "HTTP://serveurA/root/default")]
19    roleB = "classB.key=b" ;
20 } ;
```

FIG. 22 – *Utilisation des clauses #pragma*

Cet exemple de fichier MOF illustre l'opération de migration de l'information schématisée dans la figure 21. Dans ce fichier, la directive `namespace` (ligne 1) désigne l'espace de nommage dans lequel les spécifications seront chargées dans la plateforme destination. La directive `source` (ligne 12) désigne la plate-forme dont sont originaires les instances chargées. Le qualifieur `NonLocal` (ligne 18) utilisé ici à la place de la directive de même

nom, désigne la plate-forme implantant l'instance référencée (l'instance de la classe B) et qui n'a pas été chargée par la plate-forme destination.

4.6 Les types de données

MOF définit un ensemble de types de données de base que peuvent prendre les attributs et paramètres des méthodes. Ces types sont donnés dans le tableau 1.

Ces types peuvent être affinés en utilisant des qualifieurs lors de la spécification d'une propriété, un retour d'une méthode ou un paramètre. Par exemple, le DMTF a définit un qualifieur `Counter` qui permet, associé à une propriété de type entier non signé, d'indiquer que la valeur ne peut qu'augmenter et ne passera à zéro que lorsque la taille maximale du compteur est atteinte.

Les propriétés ou paramètres de type vecteur sont aussi autorisés dans les spécifications CIM. Un vecteur étant défini comme une séquence de données d'un même type simple. Les vecteurs peuvent être de taille fixe ou variable. La définition d'un vecteur peut elle aussi être affinée à l'aide des qualifieurs. Le DMTF définit par exemple un qualifieur `ArrayType` permettant de distinguer entre un vecteur générique (`Bag`), odronné (`Ordered` : les éléments sont ordonnés) ou indexé (`Indexed` : les éléments sont ordonnés et invariablement indexés).

Type	Description
(u/s)int**xx**	entier signé (*sint*) ou non (*uint*). Les valeurs **xx** indiquent la taille de l'entier. Celle-ci peut être de *8*, *16*, *32* ou *64* bits.
real**xx**	réel pouvant être sur *32* ou *64* bits. Le codage est un codage IEEE.
boolean	booléen
char16	caractère sur 16 bits codage UCS-2.
string	chaîne de caractères UCS-2.
datetime	Format de date: `yyyymmddhhmmss.mmmmmmsutc` ou `yyyy` représente l'année, `mm` les mois, `dd` le jour, `hh` l'heure courante, `mm` les minutes, `ss` les secondes, `mmmmmm` les microsecondes, `s` le signe du décalage (`+` ou `-`) suivi de l'offset `utc` exprimé en minutes. `s` peut prendre la valeur : pour spécifier un intervalle de temps.

TAB. 1 – *Les types de données primitifs de MOF*

4.7 La notation graphique

Dans les pages précédentes, nous avons présenté la notation textuelle du formalisme MOF. Les concepteurs de l'approche ont également associé à cette notation, un formalisme graphique issu du langage UML [9] (Unified Modeling Language). Les composants graphiques de la notation MOF sont donnés dans la figure 23.

On retrouve dans la notation graphique un composant pour la description d'une classe, d'une instance ainsi qu'un certain nombre de composants graphiques pour la description des relations entre objets. Ces descripteurs permettent de représenter des relations :

- d'héritage ;
- de référence ;
- d'associations simples (différentes cardinalités) ;
- d'associations comportant des informations propres.

Si cette notation permet de spécifier des schémas MOF, il n'existe pas de support actuel pour modéliser les espaces de nommages dans la notation graphique. Cela ne pose pas vraiment de problème car les espaces de nommages sont eux mêmes modélisés par des objets dans l'approche CIM.

4.8 Résumé

Dans cette section, nous avons présenté le langage MOF défini dans l'approche WBEM pour la spécifiation des ressources gérées. Basé sur un ensemble de règles syntaxiques simples et sur un nombre de composants réduit (classe, attribut, qualifieur, méthode, instance, référence), il permet de rapidemment spécifier des modèles de l'information de gestion.

9. http://www.rational.com

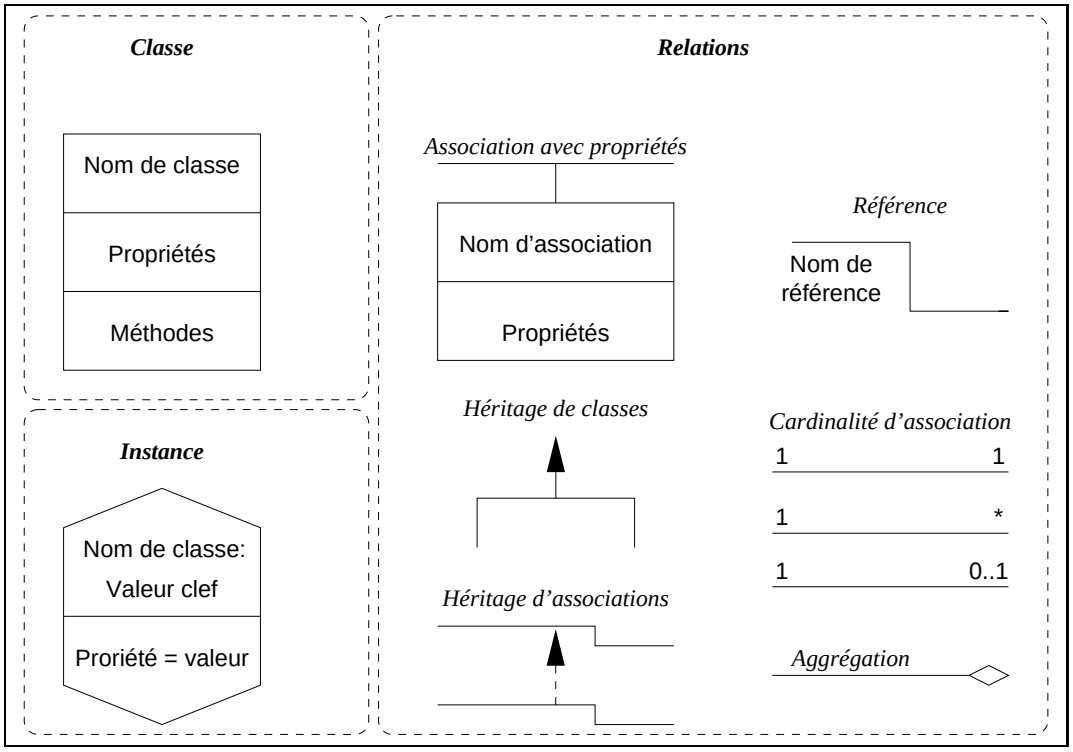


FIG. 23 – *Les composants graphiques de la notation MOF*

Le couplage fort avec le langage graphique de classes d'UML lui ouvre le support de multiples environnements de conception et de développement, permettant un essor et une acceptation plus facile du langage dans les entreprises [10].

Cela dit, la question rémmanante lors du développement d'une nouvelle approche de gestion est *fallait-il concevoir un nouveau langage pour la description des modèles de l'information?*

En effet, on se rend vite compte que MOF n'apporte que peu d'éléments supplémentaires (instances, qualifieurs) à GDMO/GRM (Guidelines for the Definition of Managed Objects / General Relationship Model). Une version simplifiée de ce langage aurait probablement suffit. De plus, si le langage est, il faut l'avouer, assez simple à assimiler, le fait de déléguer une grande partie de la sémantique des composants aux qualifieurs, n'est certainement pas la meilleure approche qui soit. Certains qualifieurs auraient très bien pu être intégrés comme composants du langage (Association, Indication, Key).

5 Les schémas du modèle commun

Afin d'unifier la vision des ressources gérées, le DMTF a défini 2 schémas de base, assez génériques, sur lesquels devrait s'appuyer toute modélisation du système géré. Le premier est appelé schéma de base (*Core Schema*). Il comporte une modélisation minimale de l'environnement géré. Le second, appelé schéma commun (*Common Schema*) étend le schéma de base sur un certain nombre de domaines.

Nous allons dans la suite de cette section présenter brièvement chacun de ces schémas. Pour une spécification complète de ces schémas, nous renvoyons le lecteur au site du DMTF [11].

5.1 Le schéma de base

Le schéma de base définit une classification générale des éléments de l'environnement géré. Ce schéma générique doit servir de référence à tous les modèles futurs développés dans le cadre de l'approche. Pour cette raison, il définit un nombre minimal de classes et de relations et a pour objectif de modéliser de manière très

10. C'est notamment l'abscence d'outils qui a fortement freiné le développement de GDMO (Guidelines for the Definition of Managed Objects) qui cherche toujours une notation graphique associée et un couplage avec OMT (Object Modeling Technique), UML ou tout autre formalisme graphique.

11. `http://www.dmtf.org`

abstraite l'ensemble des éléments gérés qui sont indépendents de tout domaine d'application. Ce modèle est très stable et ne devrait subir que de minimes modifications éventuelles.

5.1.1 Les classes et les associations

La figure 24 comporte le schéma UML définissant le modèle de base simplifié (sans attributs ni méthodes). Les composants principaux spécifiés dans le modèle sont :

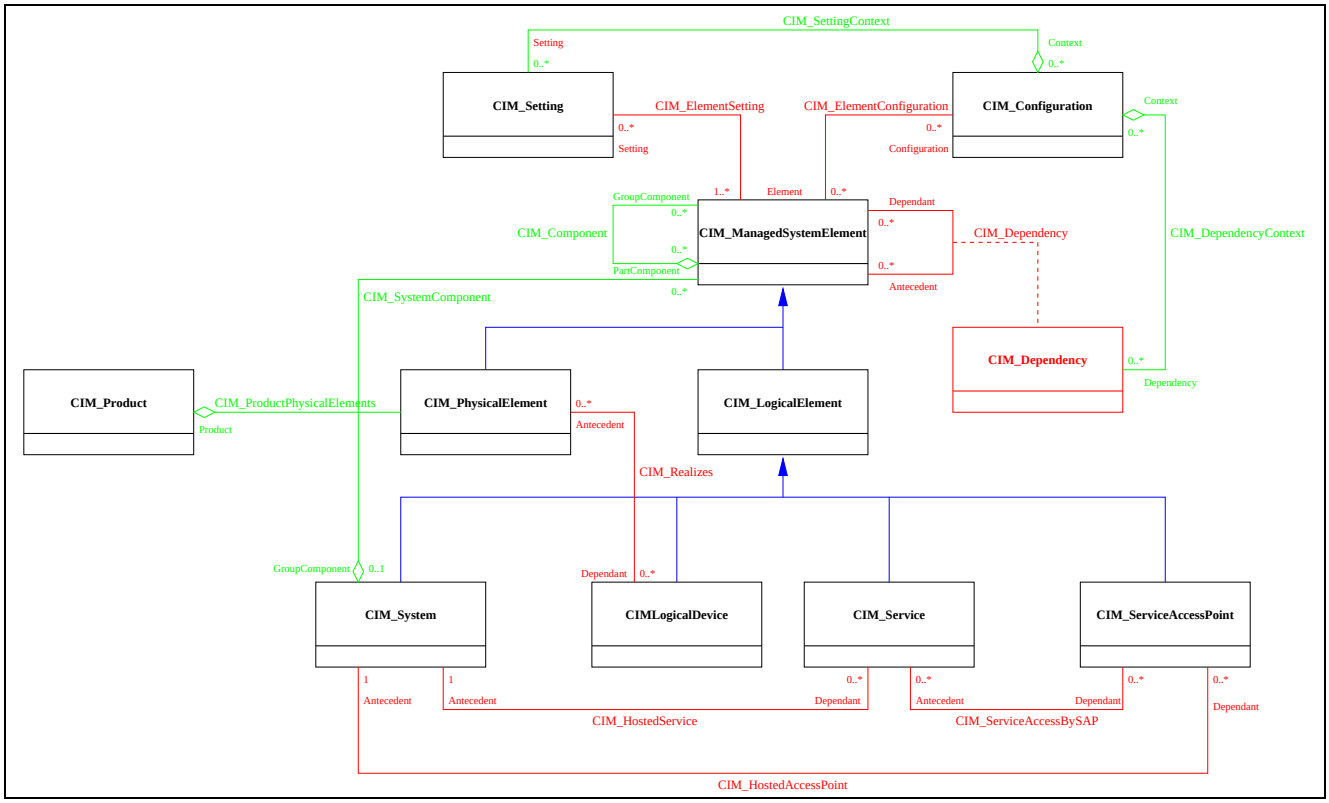


FIG. 24 – *Le noyau du modèle de l'information*

1. Un élément géré (`CIM_ManagedSystemElement`) est une classe abstraite définie pour modéliser tout composant, au sens système, intéressant du point de vue de la gestion. Le modèle distingue deux types d'éléments gérés : l'élement physique (`CIM_PhysicalElement`) qui est une représentation d'un composant physique, i.e. une réalité matérielle[12], et l'élément logique (`CIM_LogicalElement`) qui est une abstraction, utilisée pour la gestion, configuration et coordination, de l'environnement physique ou logiciel du système. Par exemple, une station de travail sera modélisée par un composant physique (caisson, localisation dans un bureau, ...) mais pourra également être modélisée par un élément logique qui en reflètera certaines caractéristiques (état administratif, état de fonctionnement, ...) ;
2. Un service (`CIM_Service`) est une représentation, dans le contexte de la gestion et de la configuration, de l'implémentation d'une fonctionnalité offerte par un ou plusieurs systèmes ;
3. Un point d'accès au service (`CIM_ServiceAccesspoint`) est une abstraction pour la gestion, la quantification et la configuration de l'accès à un service donné ;
4. Un produit (`CIM_Product`) est une représentation d'un contrat entre un vendeur et un consommateur. Il permet notamment de donner les caractéristiques d'un produit telles que sa date d'acquisition, sa maintenance ou encore son installation ;
5. Une association de dépendance (`CIM_Dependency`) permet de modéliser des relations de dépendances entre les objets gérés. On distingue deux types de dépendances :
   - la dépendance existancielle signifie qu'un élément ne peut pas exister sans un autre élément, telle que la dépendance qui existe entre un service donné et le système qui l'implante (`CIM_HostedService`) ;

12. La définition normative précise que ce composant obéit aux lois élémentaires de la physique.

- la dépendance fonctionnelle signifie qu'un élément ne peut pas fonctionner sans un autre élément, telle que la dépendance qui existe entre un service donné et le point d'accès nécessaire pour ce service (`CIM_ServiceSAPDependency`).

6. Une aggrégation de composition (`CIM_Component`) permet de modéliser une relation de composition entre deux éléments gérés, i.e. modéliser un élément comme la composition d'un ou de plusieurs autres éléments. L'aggrégation `CIM_SystemComponent` sous-classe de `CIM_Component` permet par exemple de modèliser un système géré à l'aide des différents éléments physiques qui le composent.

5.1.2 Extension du modèle

Bien que le modèle de base soit stable, celui-ci ne permet pas de dériver (par spécialisation) toutes les modélisations possibles et immaginables. Par exemple, il n'est pas possible à partir du modèle de base de dériver une classe modélisant un utilisateur. Pour cela, il est possible que le modèle de base puisse évoluer et incorporer de novelles classes. Cependant, cette évolution ne devrait en rien modifier les classes et dépendances existantes. Comme il est par ailleurs possible de définir de nouveaux modèles dont certains objets ne sont pas dérivés du modèle de base, il n'est pas forcémment nécessaire de le faire évoluer systématiquement. Une évolution ne pourra se faire qu'au travers d'un consensus sur les composants à rajouter en respectant des règles strictes garantissant au modèle son caractère universel.

5.2 Le schéma commun

Le schéma commun est une extension du schéma de base. Il rassemble un ensemble de spécifications qui paraissent utiles à un grand nombre d'applications. Dans la version actuelle du modèle CIM, le schéma commun présente des modèles de l'information pour de nombreux composants de gestion. Ces modèles ont été présentés pour la première fois au congrès annuel du DMTF en 1999 mais les spécifications détaillées ne sont pour le moment accessibles qu'aux membres du consortium. Les modèles sont les suivants :

- Le modèle système : permet de modéliser de manière générique un système distribué. Il comprend des objets qui modélisent un oridnateur isolé, un groupe d'ordinateurs (cluster) qui fonctionnent ensemble, des objets de virtualisation ou émulation d'un système ou d'un cluster et bien sûr tous les composants logiques d'un système (service de boot, système d'exploitation, processus, processeur, mémoire, thread, système de fichier, bios, tâche, ...). Ce modèle est l'un des modèles les plus complets à ce jour ;
- Le modèle physique : permet de modéliser la composition physique des éléments gérés. Il spécifie en terme d'objets gérés tous les composants physiques d'un système : chassis, rack, transformateur, disque, RAM, slots, ...). Ce modèle, à l'image du modèle système est aujourd'hui complet ;
- Le modèle de device : représente une virtualisation des services réalisés par des devices ainsi que les dépendances entre ces devices pour la réalisation d'un service. En plus des devices de base (disque, controleur, ...), ce modèle complète les autres modèles de CIM par la spécification de deux principaux types de dépendances: redondance et maître/esclave. Ces relations sont utiles principalement dans des modèles plus spécifiques de devices tels que les devices de sauvegarde, USB ou les imprimantes;
- Spécialisation du modèle de device, le modèle de sauvegarde représente sous forme d'objets gérés tous les composants qui entrent en jeu dans ce service: les controleurs (parallèle, série, SCSI, IDE, PCI, PCMCIA, ...), les interfaces (série, SCSI, ...), les supports (disque dur, disquette, DVD, CDROM, Cassette, ...) ainsi que l'organisation des volumes (partitionnement des disques, disque logique, volume de sauvegarde, ...). Ce modèle permet de représenter la chaîne de dépendance pour tous les types et organisations de sauvegarde ;
- Seconde spécialisation du modèle de device, la définition des objets gérés pour PCI et USB fournit une spécialisation de l'objet controleur pour les BUS PCI et USB. Ce modèle se focalise sur la partie bus d'un device et décrit toutes les relations permettant de relier les composants ;
- dernière spécialisation du modèle de device, la spécification du modèle d'imprimantes définit tous les objets qui interviennent dans un service d'imprqession (imprimante, file d'impression, tâches d'impression, point d'accès, protocole utilisé, service offert: type de papier, type de format supporté, ...) ;
- Le modèle application : permet de modéliser une application distribuée de type client/serveur en identifiant ses différents composants logiciels. Les composants de base du modèle sont le système applicatif dans son ensemble, ses caractéristiques, ses composants en terme d'unités de déploiement ainsi que les liaisons avec les objets produit et services pour les dépendances fonctionnelles. Le modèle fournit également une

caractérisation d'état d'application distribuée (déployable, installable, exécutable, en cours d'exécution) ainsi qu'un modèle condition/action décrivant les transitions possibles entre ces états ;

- le modèle de performance d'applications distribuées complète le modèle d'application en définissant les objets gérés qui décrivent les métriques pour tester le fonctionnement d'applications distribuées. L'objet de base de ce modèle est l'unité de traitement qui modélise un traitement en cours (impression, une transaction, une tâche en batch, une sauvegarde, ...) ou un traitement terminé. Cet objet est lié à un objet de définition de métrique et comporte des mesures propres au traitement en cours et/ou terminé tels que le temps de réponse, la durée, et d'autres informations spécifiées dans la définition de la métrique ;
- Le modèle de communication spécifie les adaptateurs réseau. Ce modèle est aujourd'hui un modèle spécifique pour des MODEM et comprend 2 spécialisations: une orientée connexion pour les modems cables et l'ADSL; une seconde pour les interactions orientées appels tels quel la téléphonie classique ou le RNIS;
- le modèle d'utilisateurs a pour objectifs de fournir l'ensemble des objets gérés nécessaires à la modélisation des utilisateurs. L'objet de base est le `party` qui représente une personne dans le contexte d'une hiérarchie organisationnelle. Le modèle sépare la personne de l'organisation et offre un mécanisme de groupage contextuel. Les informations modélisées sont des informations d'adresse, de comptes, de modèles d'authentification et de relations de groupes;
- le modèle de sécurité complète le modèle utilisateur avec une spécification de la notion de compte et tous les objets liés à la sécurité et à l'authentification des usagers (clefs publiques, privées, partagées, authorité de confiance, gestion de comptes utilisateurs, service de gestion de l'authentification, des autorisations, ...). Une spécialisation de ces objets dans le contexte Kerberos est également disponible dans le modèle.

Ce modèles forment une base pour la gestion système. Deux modèles supplémentaires sont en cours de standardisation et ne rentrent pas dans ce schéma de supervision de système. C'est pourquoi nous les avons séparés des autres. Ces modèles sont le modèle réseau et le modèle de politiques. Le modèle réseau est important car il ouvre le champ d'application de CIM au domaine de la gestion de réseaux, ce qui n'était pas ou peut-être pas son domaine cible initialement.

Le modèle de politiques ammène encore plus WBEM dans le monde de la gestion des réseaux et services. En effet, toute approche de supervision qui disposera d'un tel modèle le plus rapidemment possible pourra devenir un standard de fait dans le monde aujourd'hui désert des architectures pour la gestion de services. Les partenaires du DMTF, notamment CISCO et Microsoft ont parfaitement analysé ce besoin, notamment dans le monde IP de demain avec la différenciation de services, et des efforts importants, en lien avec les travaux à l'IETF sont mis en œuvre pour un tel modèle.

Finalement, le modèle réseau ainsi que le modèle de politiques forment la passerelle entre l'approche WBEM et l'approche DEN (Directory Enabled Networks) [22] fortement encouragée par de nombreux constructeurs d'équipements notamment CISCO et 3COM.

5.3 Les schémas supplémentaires

Il est bien sûr possible pour un concepteur d'un système de gestion de définir son propre modèle de l'information (schéma). Cependant, tout schéma doit prendre en compte les composants du schéma général et s'appuyer sur le schéma commun. Cela a pour but de maintenir une vision uniforme des ressources gérées.

Les schémas supplémentaires sont fournis par les constructeurs de machines ou de logiciels. Dans la section présentant les implémentations actuelles de WBEM (sec. 9), nous présentons deux exemples de ces shémas supplémentaires : le schéma `Solaris` (sec. 9.2.6) utilisé par l'implémentation de SUN Microsystms pour la gestion des systèmes Solaris, et le schéma `Win32` (sec. 9.1.5) utilisé par l'implémentation de Microsoft pour la gestion des environnements Windows.

5.4 Résumé

Conscients du besoin de disposer d'un schéma commun entre toutes les applications de gestion afin de permettre une vision uniforme de ses ressources, deux modèles successifs ont été développés pour former une base de spécification des ressources gérées. D'une part, le modèle de base définit un nombre minimal de classes et de dépendances entre ces classes. D'autre part, le modèle commun étend le modèle de base sur un certain nombre de domaines. Finalement, les modèles supplémentaires permettent aux constructeurs d'enrichir les modèles communs avec des objets spécifiques à leurs systèmes. Comme nous l'avons vu via l'enumération des modèles existants, ceux-ci sont nombreux et la plupart ont aujourd'hui atteint une maturité suffisante pour leur déploiement.

6 Le modèle de communication

La définition d'un modèle de communication entre les différentes entités de gestion d'une architecture WBEM a jusqu'il y a quelques mois été le maillon faible de l'approche. En effet, après la disparition du protocole HMMP (Hyper-Media Management Protocol) initialement prévu pour véhiculer l'information CIM [13], la définition d'un modèle de communication entre entités WBEM a été laissée au choix de l'implémentation [14].

Dernièrement [15], le DMTF a surmonté cette lacune en proposant un couplage entre deux standards du marché du Web afin de transporter l'information CIM d'une entité WBEM vers une autre. Le premier de ces standards est le langage XML (eXtensible Markup Langage) [26] utilisé pour l'encodage de l'information. Le second est le protocole HTTP (Hyper-Text Transport Protocol) [15] utilisé pour le transport propement dit.

En plus ce couplage XML/HTTP, le DMTF a proposé un ensemble d'opérations de gestion, appelées **opérations CIM**, afin de permettre l'accès ainsi que la manipulation de données CIM.

Dans cette section nous présentons chacun de ces trois points. Dans un premier temps (sec. 6.1), nous classifions les opérations CIM qui peuvent être invoquées sur une entité WBEM. Ces opérations sont détaillées dans les sections 6.2, 6.3, 6.4, 6.5, 6.6, 6.7 et 6.8.

Ensuite (sec. 7.1), nous présentons l'encodage XML des données CIM. Enfin (sec. 7.7), nous décrivons l'encapsulation de l'information dans le protocole HTTP.

6.1 L'interface de communication

Une opération CIM est définie comme un appel de méthode sur un objet géré. Le DMTF distingue deux familles d'opérations : les opérations *intrinsèques*, et les opérations *extrinsèques*.

- Les opérations *intinsèques* correspondent aux appels de méthodes sur des objets particuliers : les espaces de nommages [16]. Notons à cet effet que dans une implémentation WBEM, l'espace de nommage est considéré comme un objet géré, instance de la classe système `__Namespace`. Ainsi, en plus de son rôle de définition du contexte d'implémentation et de nommage dans la base de gestion des objets gérés CIM (classes et instances), l'espace de nommage permet d'offrir un point d'accès aux objets qu'il contient via les méthodes *intrinsèques* telles que `getClass` ou `getInstance`, que nous détaillons dans cette section. Conceptuellement, les opérations *intrinsèques* correspondent aux services de gestion classiques du type `M-GET` ou `M-SET` dans CMIS.
- Les opérations *extrinsèques* correspondent aux appels de méthodes sur tout autre type d'objet géré CIM. Ces méthodes sont celles données dans les définitions des classes CIM (voir sec. 4), telle que la méthode `Reboot` de la classe `CIM_ComputerSystem` donnée dans l'exemple de la figure 9. Le protocole de communication XML/HTTP utilisé dans WBEM, défini un service particulier permettant d'appeler les méthodes *extrinsèques* du modèle CIM implanté. Ce service est détaillé dans la section 7.6. Ce type d'opération est l'équivalent du service M-Action dans le protocole CMIS de l'OSI.

Les opérations *intrinsèques*, définissant le noyau des services de gestion WBEM, sont classifiées en sept profils différents, correspondant chacun à une logique fonctionnelle précise. À ces profils sont associés des relations de dépendance vis-à-vis de l'implantation : un serveur CIM ne peut implanter les opérations d'un profil donné que s'il implante celles des profils dont il dépend [17]. Ces profils sont les suivants :

1. Les opérations de lecture de base (**Basic Read**) ;
2. Les opérations d'écriture de base (**Basic Write**) ;
3. Les opérations de manipulation de schéma (**Schema Manipulation**) ;
4. Les opérations de manipulation d'instances (**Instance Manipulation**) ;
5. Les opérations de parcours d'associations (**Association Traversal**) ;
6. Les opérations de requêtes (**Query Execution**) ;
7. Les opérations de déclaration de qualifieurs (**Qualifiers Declaration**) ;

13. mais qui lui-même ne s'appuyait sur aucun protocole de transport connu!
14. Ce qui a pour conséquence naturelle de stopper net toute tentative d'interopérabilité!
15. Depuis le 11 août 1999.
16. Dans la suite, nous utilisons l'appellation "espace de nommage cible" pour désigner l'espace de nommage sur lequel est appliqué une méthode *intrinsèque*
17. Dans la norme, la définition de l'implantation d'un profil est donnée sous la forme d'une négation : si un profil n'est pas supporté alors aucune opération de ce profil n'est supportée !

Dans les sections qui suivent, nous détaillons les opérations définies dans chacun de ces profils. Les notations utilisées pour décrire les opérations sont celles données dans la norme. Elles correspondent à une notation comparable à celle utilisée dans MOF pour la spécification d'une méthode dans une classe CIM (voir section 4). Les qualifieurs, associés aux paramètres des méthodes, sont purement descriptifs, et permettent de donner la sémantique de ces derniers.

- Le qualifieur `IN` désigne un paramètre d'entrée ;
- le qualifieur `OPTIONAL` désigne un paramètre optionnel lors de l'appel de la méthode ;
- le qualifieur `NULL` désigne un paramètre qui peut avoir une valeur nulle lors de l'appel de la méthode.

6.2 Les opérations de lecture de base

Ce domaine fonctionnel regroupe toutes les opérations de lecture simple sur les données CIM maintenues par un serveur, tel que la demande de définition d'une classe ou d'une instance dans l'espace de nommage cible.

Ce profil est le seul indépendant de tout autre profil. Tout serveur CIM doit supporter au moins un sous-ensemble des opérations qui le définissent.

6.2.1 L'opération GetClass

L'opération **GetClass** permet à un client de demander à un serveur de lui retourner la définition d'une classe implantée dans l'espace de nommage cible. Les paramètres de l'opération sont donnés dans la figure 25 :

```
1 <class>  GetClass (
2     [IN] <className>  ClassName,
3     [IN,OPTIONAL] boolean  LocalOnly = true,
4     [IN,OPTIONAL] boolean  IncludeQualifiers = true,
5     [IN,OPTIONAL] boolean  IncludeClassOrigin = true,
6     [IN,OPTIONAL,NULL] string  PropertyList[] = NULL
7 )
```

FIG. 25 – *L'opération GetClass*

Le paramètre `ClassName` (ligne 2) permet au client de spécifier le nom de la classe dont il désire recevoir la définition. Le type `className` correspond à l'élément XML `CLASSNAME` (voir sec. 7.1), et permet de donner un nom de classe. Ceci représente le champ `ModelPath` défini dans le nommage d'une classe CIM (voir sec. 3.2).

Le paramètre `LocalOnly` (ligne 3) est optionnel et constitue un filtre pour l'opération. Ce filtre limite la réponse aux seuls éléments (propriétés, méthodes ou qualifieurs) explicitement définis dans la classe sélectionnée, excluant ainsi tous les éléments issus des mécanismes d'héritage. Si ce paramètre est fixé à `true` seuls les éléments définis dans classe seront retournés dans la réponse. Dans le cas contraire, la réponse comprendra ceux-ci ainsi que tous les éléments hérités par la classe. Par défaut, la valeur `true` est attribuée au paramètre.

Le paramètre `IncludeQualifiers` (ligne 4) est optionnel et constitue un filtre pour l'opération. Si le filtre est fixé à `false`, aucun qualifieur associé à la classe ou à l'une de ses propriétés, méthodes ou références n'est retourné dans la réponse, sinon tous les qualifieurs sont inclus dans la réponse. Par défaut, la valeur `true` est attribuée au paramètre.

Le paramètre `IncludeClassOrigin` (ligne 5) est optionnel et constitue un filtre pour l'opération. Si le filtre est fixé à `false`, aucune clause CLASSORIGIN (elle permet de désigner la classe contenant la définition originelle d'un attribut ou une méthode de la classe - voir sec. 7.2.1) n'est inclue dans la réponse, sinon toutes les clauses CLASSORIGIN sont retournées. Par défaut, la valeur `false` est attribuée au paramètre.

Le paramètre `PropertyList` (ligne 6) est optionnel et constitue un filtre pour l'opération. Ce filtre permet de spécifier une liste de noms d'attributs et limite la réponse aux seuls attributs de la classe donnés dans cette liste. Si la liste donnée est la liste vide, aucune propriété ne sera retournée dans la réponse. Si le paramètre est fixé à `NULL`, toutes les propriétés (sélectionnées en fonction des autres filtres de l'opération) seront retournées. Par défaut, la valeur `NULL` est attribuée au paramètre.

En cas de succès, l'opération retourne une définition de classe. Le type de retour `class` correspond à l'élément XML `CLASS` (voir sec. 7.1), et permet de donner une représentation XML de la définition d'une classe CIM.

En cas d'échec, un code d'erreur est renvoyé par le serveur. Les codes d'erreurs possibles sont :

- CIM_ERR_ACCESS_DENIED : l'accès à l'information n'est pas autorisé ;

- CIM_ERR_INVALID_NAMESPACE : l'espace de nommage cible n'existe pas ;
- CIM_ERR_INVALID_PARAMETER : l'un des paramètres de l'opération est mal défini ;
- CIM_ERR_NOT_FOUND : la définition requise n'a pas été trouvée dans l'espace de nommage cible ;
- CIM_ERR_FAILED : l'opération n'a pas pu aboutir pour une raison non spécifiée.

6.2.2 L'opération EnumerateClasses

L'opération **EnumerateClasses** permet à un client de demander à un serveur de lui renvoyer un ensemble de définitions de classes. Les critères de sélection sont les suivants :

- demande de l'ensemble des sous-classes d'une classe donnée que nous désignerons dans la suite par classe de base de la requête ;
- demande de l'ensemble des classes définies dans l'espace de nommage cible.

Les paramètres de l'opération sont donnés dans la figure 26.

```
1 <class>* EnumerateClasses (
2     [IN,OPTIONAL,NULL] <className>  ClassName = NULL,
3     [IN,OPTIONAL] boolean  DeepInheritance = true,
4     [IN,OPTIONAL] boolean  LocalOnly = true,
5     [IN,OPTIONAL] boolean  IncludeQualifiers = true,
6     [IN,OPTIONAL] boolean  IncludeClassOrigin = true
7 )
```

FIG. 26 – *L'opération EnumerateClasses*

Le paramètre `ClassName` (ligne 2) est optionnel et permet de donner le nom de la classe à la base de l'énumération. Si ce champ est non nul, seules les sous-classes de la classe de base sont sélectionnées, sinon (champ non rempli, ou valeur `NULL`) toutes les classes de l'espace de nommage sont retournées. Par défaut, la valeur `NULL` est attribuée au paramètre.

Le paramètre `DeepInheritence` (ligne 3) est optionnel et permet d'affiner le critère de sélection des classes à retourner au client en précisant la portée de la requête. Si la classe de base est spécifiée (champs `ClassName`), et si ce champs est fixé à `false` seules les sous-classes directes (premier niveau dans l'arbre d'héritage) de la classe de base sont sélectionnées. Sinon toutes les sous-classes sont retournées. Si aucune classe de base n'est spécifiée, et si ce champs est fixé à `false`, seules les super-classes (les classes qui n'héritent d'aucune autre classe) implantées dans l'espace de nommage sont sélectionnées, sinon toutes les classes de l'espace de nommage sont retournées. Par défaut, la valeur `true` est attribuée au paramètre.

Les champs `LocalOnly`, `IncludeQualifiers` et `IncludeClassOrigin` (lignes 4, 5 et 6) sont optionnels et forment des filtres pour l'opération. Ils sont identiques à ceux définis pour l'opération **GetClass**.

En cas de succès, l'opération retourne la liste des classes sélectionnées. En cas d'échec, un code d'erreur est renvoyé par le serveur. Les codes d'erreurs possibles sont :

- CIM_ERR_ACCESS_DENIED : l'accès à l'information n'est pas autorisé ;
- CIM_ERR_INVALID_NAMESPACE : l'espace de nommage cible n'existe pas ;
- CIM_ERR_INVALID_PARAMETER : l'un des paramètres de l'opération est mal défini ;
- CIM_ERR_INVALID_CLASS : la classe de base de la requête n'a pas été trouvée dans l'espace de nommage cible ;
- CIM_ERR_NOT_SUPPORTED : l'opération n'est pas implantée par le serveur ;
- CIM_ERR_FAILED : l'opération n'a pas pu aboutir pour une raison non spécifiée.

6.2.3 L'opération EnumerateClassNames

L'opération **EnumerateClassNames** est une version allégée de l'opération `EnumerateClasses`. Elle permet à un client de demander à un serveur de sélectionner un ensemble de classes selon les mêmes critères que ceux de l'opération `EnumerateClasses`, mais de ne lui renvoyer que les noms des classes sélectionnées. Les paramètres de l'opération sont donnés dans la figure 27.

```
1 <className>* EnumerateClassNames (
2     [IN,OPTIONAL,NULL] <className> ClassName = NULL,
3     [IN,OPTIONAL] boolean DeepInheritance = true
4 )
```

FIG. 27 – *L'opération EnumerateClassNames*

Les champs `ClassName` et `DeepInheritence` (lignes 2 et 3) sont identiques à ceux définis dans l'opération `EnumerateClasses`.

En cas de succès, l'opération retourne la liste des noms de classes sélectionnées. En cas d'échec, un code d'erreur est renvoyé par le serveur. Les codes d'erreurs possibles sont identiques à ceux de l'opération **EnumerateClasses**.

6.2.4 L'opération GetInstance

L'opération **GetInstance** permet à un client de demander à un serveur de lui renvoyer la définition d'une instance dans l'espace de nommage cible. Les paramètres de l'opération sont donnés dans la figure 28.

```
1 <instance> GetInstance (
2     [IN] <instanceName> InstanceName,
3     [IN,OPTIONAL] boolean LocalOnly = true,
4     [IN,OPTIONAL] boolean IncludeQualifiers = true,
5     [IN,OPTIONAL] boolean IncludeClassOrigin = true,
6     [IN,OPTIONAL,NULL] string PropertyList[] = NULL
7 )
```

FIG. 28 – *L'opération GetInstance*

Le paramètre `InstanceName` (ligne 2) permet au client de désigner le nom de l'instance dont il souhaite recevoir la définition. Le type `instanceName` correspond à l'élément XML `INSTANCENAME` (voir sec. 7.1), et permet de donner un nom d'instance, i.e. le nom de la classe suivi de la liste des couples <attibut clef / valeur>. Ceci représente le champ `ModelPath` défini dans le nommage d'une instance CIM (voir sec. 3.2).

Les autres paramètres forment des filtres pour l'opération et sont identiques à ceux définis dans l'opération **GetClass**.

En cas de succès, l'opération retourne la définition de l'instance désignée. Le type de retour `instance` correspond à l'élément XML `INSTANCE` (voir sec. 7.1), et permet de donner une représentation XML de la définition d'une instance CIM.

En cas d'échec, un code d'erreur est renvoyé par le serveur. Les codes d'erreurs possibles sont les suivants :

- CIM_ERR_ACCESS_DENIED : l'accès à l'information n'est pas autorisé ;
- CIM_ERR_INVALID_NAMESPACE : l'espace de nommage cible n'existe pas ;
- CIM_ERR_INVALID_PARAMETER : l'un des paramètres de l'opération est mal défini ;
- CIM_ERR_INVALID_CLASS : la définition de la classe de l'instance n'a pas été trouvée dans l'espace de nommage cible ;
- CIM_ERR_NOT_FOUND : la définition de la classe existe, mais l'instance requise n'a pas été trouvée dans l'espace de nommage cible ;
- CIM_ERR_FAILED : l'opération n'a pas pu aboutir pour une raison non spécifiée.

6.2.5 L'opération EnumerateInstances

L'opération **EnumerateInstances** permet à un client de demander à un serveur l'ensemble des instances d'une classe donnée dans l'espace de nommage cible. Les paramètres de cette opération sont donnés dans la figure 29.

Le paramètre `ClassName` (ligne 2) permet de désigner la classe de base dont les instances sont requises par le client.

```
1 <namedInstance>* EnumerateInstances (
2     [IN] <className> ClassName,
3     [IN,OPTIONAL] boolean DeepInheritance = true,
4     [IN,OPTIONAL] boolean LocalOnly = true,
5     [IN,OPTIONAL] boolean IncludeQualifiers = true,
6     [IN,OPTIONAL] boolean IncludeClassOrigin = true,
7     [IN,OPTIONAL,NULL] string PropertyList[] = NULL
8 )
```

FIG. 29 – *L'opération EnumerateInstances*

Le paramètre `DeepInheritance` (ligne 3) permet d'affiner le critère de sélection en précisant la portée de l'opération. Si ce champs est fixé à `false`, seules les instances directes de la classe de base sont sélectionnées, sinon toutes les instances de la classe ainsi que celles ses sous-classes sont retournées par le serveur.

Les autres paramètres forment des filtres pour l'opération et sont identiques à ceux définis dans l'opération `GetClass`.

En cas de succès, l'opération retourne la liste des définitions des instances sélectionnées. Le type de retour `nameInstance` correspond à l'élément XML `VALUE.NAMEDINSTANCE` (voir sec. 7.1). Ceci permet de donner la représentation XML de la définition d'une instance (élément `INSTANCE`), précédée par la représentation XML du nom de l'instance (élément `INSTANCENAME`).

En cas d'échec, un code d'erreur est renvoyé par le serveur. Les codes d'erreurs possibles sont identiques à ceux de l'opération **EnumerateClasses**.

6.2.6 L'opération EnumerateInstanceNames

L'opération **EnumerateInstanceNames** est une variante de l'opération `EnumerateInstances`. Elle permet à un client de demander à un serveur de sélectionner l'ensemble des instances d'une classe donnée dans l'espace de nommage cible, mais de ne lui renvoyer que les noms de ces instances. La définition de l'opération est donnée dans la figure 30.

```
1 <instanceName>* EnumerateInstanceNames (
2     [IN] <className> ClassName
3 )
```

FIG. 30 – *L'opération EnumerateInstanceNames*

L'unique paramètre de l'opération `ClassName` (ligne 2) permet de désigner le nom de la classe dont les instances sont requises.

En cas de succès, l'opération retourne la liste des noms des instances sélectionnées. En cas d'échec, un code d'erreur est renvoyé par le serveur. Les codes d'erreurs possibles sont identiques à ceux de l'opération **EnumerateClassNames**.

6.2.7 L'opération GetProperty

L'opération **GetProperty** permet à un client de demander à un serveur de lui retourner la valeur prise par une propriété d'une instance donnée dans l'espace de nommage cible. Les paramètres de l'opération sont donnés dans la figure 31.

```
1 <propertyValue>? GetProperty (
2     [IN] <instanceName> InstanceName,
3     [IN] string PropertyName
4 )
```

FIG. 31 – *L'opération GetProperty*

Le paramètre `InstanceName` (ligne 2) permet au client de désigner le nom de l'instance cible de la requête.

Le paramètre `PropertyName` (ligne 3) permet au client de désigner le nom de la propriété dont il cherche la valeur.

En cas de succès, l'opération retourne la valeur prise par la propriété de l'instance désignée. Le type de retour `propertyValue` correspond à l'un des éléments XML `VALUE`, `VALUE.ARRAY` ou `VALUE.REFERENCE` (voir sec. 7.1). Ces éléments permettent de donner une représentation XML d'une valeur de propriété selon son type ; simple, vecteur ou référence vers un objet. Si la propriété de l'instance a une valeur nulle (`NULL`), aucun élément n'est retourné par le serveur.

En cas d'échec, un code d'erreur est renvoyé par le serveur. Les codes d'erreurs possibles sont les suivants :

- CIM_ERR_ACCESS_DENIED : l'accès à l'information n'est pas autorisé ;
- CIM_ERR_INVALID_NAMESPACE : l'espace de nommage cible n'existe pas ;
- CIM_ERR_INVALID_PARAMETER : l'un des paramètres de l'opération est mal défini ;
- CIM_ERR_INVALID_CLASS : la définition de la classe de l'instance n'a pas été trouvée dans l'espace de nommage cible ;
- CIM_ERR_NOT_FOUND : la définition de la classe existe, mais l'instance requise n'a pas été trouvée dans l'espace de nommage cible ;
- CIM_ERR_NO_SUCH_PROPERTY : l'instance d'objet existe, mais la propriété spécifiée n'appartient à pas la liste des atributs de l'objet géré ;
- CIM_ERR_FAILED : l'opération n'a pas pu aboutir pour une raison non spécifiée.

6.3 Les opérations d'écriture de base

Ce profil fonctionnel ne comporte qu'une unique opération qui est la modification de la valeur d'une propriété d'une instance donnée.

Au niveau de l'implémentation, le profil dépend de celui de lecture de base. Tout serveur supportant l'opération d'écriture, doit supporter les opérations de lecture simple.

6.3.1 L'opération SetProperty

L'opération **SetProperty** permet à un client de demander à un serveur d'affecter une nouvelle valeur à une propriété d'une instance donnée dans l'espace de nommage cible. Les paramètres de cette opération sont donnés dans la figure 32.

```
1 void  SetProperty (
2     [IN] <instanceName>  InstanceName,
3     [IN] string  PropertyName,
4     [IN,OPTIONAL,NULL] <propertyValue>  NewValue = NULL
5 )
```

FIG. 32 – *L'opération SetProperty*

Les paramètres `InstanceName` et `PropertyName` (lignes 2 et 3) sont identiques à ceux définis dans l'opération `GetProperty`. Ils permettent de spécifier l'instance et la propriété cibles de la requête.

Le paramètre `NewValue` est optionnel, et permet au client de désigner la nouvelle valeur à affecter à la propriété de l'instance. Par défaut, la valeur `NULL`, est attribuée au paramètre.

En cas de succès, aucun élément n'est retourné par le serveur. En cas d'échec, un code d'erreur est renvoyé par le serveur. Les codes d'erreurs possibles sont les suivants :

- CIM_ERR_ACCESS_DENIED : l'accès à l'information n'est pas autorisé ;
- CIM_ERR_INVALID_NAMESPACE : l'espace de nommage cible n'existe pas ;
- CIM_ERR_INVALID_PARAMETER : l'un des paramètres de l'opération est mal défini ;
- CIM_ERR_INVALID_CLASS : la définition de la classe de l'instance n'a pas été trouvée dans l'espace de nommage cible ;
- CIM_ERR_NOT_FOUND : la définition de la classe existe, mais l'instance requise n'a pas été trouvée dans l'espace de nommage cible ;

- CIM_ERR_NO_SUCH_PROPERTY : l'instance d'objet existe, mais la propriété spécifiée n'appartient à pas la liste des atributs de l'objet géré ;
- CIM_ERR_TYPE_MISMATCH : la valeur donnée en paramètre ne correspond pas au type de la propriété désignée ;
- CIM_ERR_FAILED : l'opération n'a pas pu aboutir pour une raison non spécifiée.

6.4 Les opérations de manipulation d'instance

Ce profil fonctionnel regroupe les opérations de création, suppression ou modification d'une instance d'objet (ou de relation) dans l'espace de nommage cible.

Au niveau de l'implémentation, il dépend du profil d'écriture de base ainsi que du profil de lecture de base par transitivité.

6.4.1 L'opération CreateInstance

L'opération **CreateInstance** permet de créer et de rajouter une nouvelle instance d'objet (ou de relation) dans l'espace de nommage cible. La définition de l'opération est donnée dans la figure 33.

```
1 <instanceName>  CreateInstance (
2     [IN] <instance>  NewInstance
3 )
```

FIG. 33 – *L'opération CreateInstance*

L'unique paramètre de l'opération `NewInstance` (ligne 2) permet de donner la définition de la nouvelle instance à rajouter dans l'espace de nommage cible.

En cas de succès, l'opération retourne le nom de l'instance créée dans la base de gestion du serveur (correspondant au champ `ModelPath` pour le nommage d'une instance CIM).

En cas d'échec, un code d'erreur est renvoyé par le serveur. Les codes d'erreurs possibles sont :

- CIM_ERR_ACCESS_DENIED : l'accès à l'information n'est pas autorisé ;
- CIM_ERR_NOT_SUPPORTED : l'opération n'est pas implantée par le serveur ;
- CIM_ERR_INVALID_NAMESPACE : l'espace de nommage cible n'existe pas ;
- CIM_ERR_INVALID_PARAMETER : l'un des paramètres de l'opération est mal défini ;
- CIM_ERR_INVALID_CLASS : la définition de la classe de l'instance n'a pas été trouvée dans l'espace de nommage cible ;
- CIM_ERR_ALREADY_EXISTS : l'instance spécifiée est déjà définie dans l'espace de nommage cible ;
- CIM_ERR_FAILED : l'opération n'a pas pu aboutir pour une raison non spécifiée.

6.4.2 L'opération ModifyInstance

L'opération **ModifyInstance** permet de modifier la définition d'une instance d'objet (ou de relation) maintenue dans l'espace de nommage cible. Les paramètres de cette opération sont donnés dans la figure 34.

```
1 void  ModifyInstance (
2     [IN] <namedInstance>  ModifiedInstance
3 )
```

FIG. 34 – *L'opération ModifyInstance*

L'unique paramètre de l'opération `ModifiedInstance` (ligne 2) permet de donner la nouvelle définition de l'instance à modifier dans l'espace de nommage cible.

En cas de succès, aucun élément n'est retourné par le serveur. En cas d'échec, un code d'erreur est renvoyé par le serveur. Les codes d'erreurs possibles sont :

- CIM_ERR_ACCESS_DENIED : l'accès à l'information n'est pas autorisé ;

- CIM_ERR_NOT_SUPPORTED : l'opération n'est pas implantée par le serveur ;
- CIM_ERR_INVALID_NAMESPACE : l'espace de nommage cible n'existe pas ;
- CIM_ERR_INVALID_PARAMETER : l'un des paramètres de l'opération est mal défini ;
- CIM_ERR_INVALID_CLASS : la définition de la classe de l'instance n'a pas été trouvée dans l'espace de nommage cible ;
- CIM_ERR_NOT_FOUND : la définition de la classe existe, mais l'instance à modifier n'a pas été trouvée dans l'espace de nommage cible ;
- CIM_ERR_FAILED : l'opération n'a pas pu aboutir pour une raison non spécifiée.

6.4.3 L'opération DeleteInstance

L'opération **DeleteInstance** permet au client de demander au serveur de supprimer une instance donnée de l'espace de nommage cible. La définition de l'opération est donnée dans la figure 35.

```
1 void  DeleteInstance (
2     [IN] <instanceName>  InstanceName
3 )
```

FIG. 35 – *L'opération DeleteInstance*

L'unique paramètre de l'opération `InstanceName` (ligne 2) permet de donner le nom de l'instance à supprimer de l'espace de nommage cible.

En cas de succès, aucun élément n'est retourné par le serveur. En cas d'échec, un code d'erreur est renvoyé par le serveur. Les codes d'erreurs possibles sont identiques à ceux de l'opération **ModifyInstance**.

6.5 Les opérations de manipulation de schéma

Ce profil fonctionnel regroupe les opérations de création, suppression ou modification d'une définition de classe (ou d'association) dans l'espace de nommage cible.

Au niveau de l'implémentation, il dépend du profil de manipulation d'instance.

6.5.1 L'opération CreateClass

L'opération **CreateClass** permet de créer et de rajouter une nouvelle classe dans l'espace de nommage cible. La définition de l'opération est donnée dans la figure 36.

```
1 void  CreateClass (
2     [IN] <class>  NewClass
3 )
```

FIG. 36 – *L'opération CreateClass*

L'unique paramètre de l'opération `NewClass` (ligne 2) permet de donner la définition de la classe à rajouter dans l'espace de nommage cible.

En cas de succès, aucun élément n'est retourné par le serveur. En cas d'échec, un code d'erreur est renvoyé. Les codes d'erreurs possibles sont les suivants :

- CIM_ERR_ACCESS_DENIED : l'accès à l'information n'est pas autorisé ;
- CIM_ERR_NOT_SUPPORTED : l'opération n'est pas implantée par le serveur ;
- CIM_ERR_INVALID_NAMESPACE : l'espace de nommage cible n'existe pas ;
- CIM_ERR_INVALID_PARAMETER : l'un des paramètres de l'opération est mal défini ;
- CIM_ERR_ALREADY_EXIST : la classe spécifiée est déjà définie dans l'espace de nommage cible ;
- CIM_ERR_INVALID_SUPERCLASS : la super-classe donnée dans la définition de la classe n'a pas été trouvée dans l'espace de nommage cible ;
- CIM_ERR_FAILED : l'opération n'a pas pu aboutir pour une raison non spécifiée.

6.5.2 L'opération ModifyClass

L'opération **ModifyClass** permet de modifier une définition de classe dans l'espace de nommage cible. La définition de l'opération est donnée dans la figure 37.

```
1 void  ModifyClass (
2     [IN] <class>  ModifiedClass
3 )
```

FIG. 37 – *L'opération ModifyClass*

Le paramètre de l'opération `ModifiedClass` (ligne 2) permet au client de donner la nouvelle définition de la classe à modifier.

Avant de procéder à une modification d'une définition de classe, le serveur doit vérifier deux familles de contraintes :

- les contraintes d'héritage : la modification de la classe doit rester cohérente vis-à-vis de la définition de ses super-classes ainsi que celles de ses sous-classes (cohérence avec les qualifieurs, propriétés et méthodes de la super-classe et des sous-classes). De plus, si l'ancienne définition spécifie un héritage (i.e. possède une super-classe), la nouvelle définition ne doit pas modifier cet héritage ;
- les contraintes d'instanciation : la modification de la classe doit rester cohérente vis-à-vis des définitions de ses instances, i.e. que les modifications sur les propriétés ne s'opposent pas aux initialisations des instances de la classe.

En cas de succès, aucun élément n'est retourné par le serveur. En cas d'échec, un code d'erreur est renvoyé par le serveur. Les codes d'erreurs possibles sont :

- CIM_ERR_ACCESS_DENIED : l'accès à l'information n'est pas autorisé ;
- CIM_ERR_NOT_SUPPORTED : l'opération n'est pas implantée par le serveur ;
- CIM_ERR_INVALID_NAMESPACE : l'espace de nommage cible n'existe pas ;
- CIM_ERR_INVALID_PARAMETER : l'un des paramètres de l'opération est mal défini ;
- CIM_ERR_NOT_FOUND : la définition de la classe à modifier n'a pas été trouvée par le serveur ;
- CIM_ERR_INVALID_SUPERCLASS : la super-classe désignée dans la définition de la classe n'est pas valide ;
- CIM_ERR_CLASS_HAS_CHILDREN : la modification de la classe est en conflit avec les définitions de ses sous-classes ;
- CIM_ERR_CLASS_HAS_INSTANCES : la modification de la classe est en conflit avec les définitions de ses instances ;
- CIM_ERR_FAILED : l'opération n'a pas pu aboutir pour une raison non spécifiée.

6.5.3 L'opération DeleteClass

L'opération permet au client de demander au serveur de supprimer une définition de classe dans l'espace de nommage cible. La définition de l'opération est donnée dans la figure 38.

```
1 void  DeleteClass (
2     [IN] <className>  ClassName
3 )
```

FIG. 38 – *L'opération DeleteClass*

L'unique paramètre de l'opération `ClassName` (ligne 2) permet au client de donner le nom de la classe à supprimer de l'espace de nommage cible.

Avant de procéder à une suppression d'une définition de classe, le serveur doit procéder (avec succès) à deux opérations préalables :

- suppression de toutes les sous-classes de la classe à supprimer. Si l'opération échoue pour l'une des sous-classes, elle doit aussi échouer pour la classe donnée ;

– suppression de toutes les instances de la classe à supprimer. Si l'opération échoue pour l'une des instances, l'opération doit aussi échouer pour la classe donnée.

En cas de succès, aucun élément n'est retourné par le serveur. En cas d'échec, un code d'erreur est renvoyé par le serveur. Les codes d'erreurs possibles sont :

– CIM_ERR_ACCESS_DENIED : l'accès à l'information n'est pas autorisé ;
– CIM_ERR_NOT_SUPPORTED : l'opération n'est pas implantée par le serveur ;
– CIM_ERR_INVALID_NAMESPACE : l'espace de nommage cible n'existe pas ;
– CIM_ERR_INVALID_PARAMETER : l'un des paramètres de l'opération est mal défini ;
– CIM_ERR_NOT_FOUND : la définition de la classe à supprimer n'a pas été trouvée par le serveur ;
– CIM_ERR_CLASS_HAS_CHILDREN : l'opération a échoué à cause d'un échec lors de la suppression d'une ou de plusieurs sous-classs de la classe ;
– CIM_ERR_CLASS_HAS_INSTANCES : l'opération a échoué à cause d'un échec lors de la suppression d'une ou de plusieurs instances de la classe ;
– CIM_ERR_FAILED : l'opération n'a pas pu aboutir pour une raison non spécifiée.

6.6 Les opérations de parcours d'association

Ce profil fonctionnel regroupe les opérations de requêtes sur des définitions ou des instances de relations entre les objets CIM (classes ou instances).
Au niveau de l'implantation, ce profil ne dépend que du profil de lecture de base.

6.6.1 L'opération Associators

L'opération **Associators** permet de retourner la liste des classes ou des instances en relation avec un objet CIM donné (classe ou instance). Les paramètres de l'opération sont donnés dans la figure 39.

```
1  <objectWithPath>*  Associators (
2      [IN] <objectName>  ObjectName,
3      [IN,OPTIONAL,NULL] <className>  AssocClass = NULL,
4      [IN,OPTIONAL,NULL] <className>  ResultClass = NULL,
5      [IN,OPTIONAL,NULL] string  Role = NULL,
6      [IN,OPTIONAL,NULL] string  ResultRole = NULL,
7      [IN,OPTIONAL] boolean  IncludeQualifiers = false,
8      [IN,OPTIONAL] boolean  IncludeClassOrigin = false,
9      [IN,OPTIONAL,NULL] string  PropertyList[] = NULL
10 )
```

FIG. 39 – *L'opération Associators*

Le paramètre `ObjectName` (ligne 2) permet de donner le nom de la classe ou de l'instance à la base du parcours des relations. Le type `objectName` correspond à l'un des éléments XML `CLASSNAME` ou `INSTANCENAME` (voir sec. 7.1), et permet de donner le nom de classe ou de l'instance de base.

Le paramètre `AssocClass` (ligne 3) est optionnel et constitue un filtre pour l'opération. S'il est non nul dans la requête, il permet de sélectionner parmi les objets en relation avec l'objet à la base de la requête, seulement ceux qui jouent un rôle dans une instance de l'association désignée ou de l'une de ses sous-classes. Par défaut, la valeur `NULL` est attribuée au paramètre.

Le paramètre `ResultClass` (ligne 4) est optionnel et constitue un filtre pour l'opération. Il permet de sélectionner parmi les objets en relation avec l'objet à la base de la requête, ceux qui appartiennent à la classe désignée (ou à l'une de ses sous-classes). Par défaut, la valeur `NULL` est attribuée au paramètre.

Le paramètre `Role` (ligne 5) est optionnel et constitue un filtre pour l'opération. Il permet de sélectionner parmi les réponses possibles, les objets en relation avec l'objet à la base de la requête quand celui-ci joue le rôle désigné. Par défaut, la valeur `NULL` est attribuée au paramètre.

Le paramètre `ResultRole` (ligne 6) est optionnel et constitue un filtre pour l'opération. Il permet de sélectionner parmi les réponses possibles, les objets en relation avec l'objet à la base de la requête jouant le rôle désigné. Par défaut, la valeur `NULL` est attribuée au paramètre.

Le paramètre `IncludeQualifiers` (ligne 7) est optionnel et constitue un filtre pour l'opération. Si le filtre est fixé à `false` aucun qualifieur associé aux objets sélectionnés n'est retourné dans la réponse, sinon tous les qualifieurs sont inclus dans la réponse. Par défaut, la valeur `false` est attribuée au paramètre.

Le paramètre `IncludeClassOrigin` (ligne 8) est optionnel et constitue un filtre pour l'opération. Si le filtre est fixé à `false`, aucune clause `CLASSORIGIN` n'est retournée dans la réponse, sinon toutes les clauses `CLASSORIGIN` sont inclues dans la réponse. Par défaut, la valeur `false` est attribuée au paramètre.

Le paramètre `PropertyList` (ligne 9) est optionnel et constitue un filtre pour l'opération. Il permet de spécifier une liste de noms d'attributs et limite les définitions des classes ou des instances séléctionnées pour la (les) réponse(s) aux seuls attributs désignés dans cette liste. Il possède la même sémantique que celui défini dans l'opération **GetClass**.

En cas de succès, l'opération retourne la liste des objets (définitions de classes ou instances) sélectionnés. Le type de retour `objectWithPath` correspond à l'élément XML `VALUE.OBJECTWITHPATH` (voir sec. 7.1). Il permet de donner une représentation XML de la définition d'un objet CIM (élément `CLASS` ou `INSTANCE`), précédée par la représentation XML du nom de l'objet `ObjectName` défini dans le nommage CIM (nom absolu de l'espace de nommage et identificateur de l'objet dans cet espace de nommage - voir sec. 3.2 - et représenté en XML par l'un des éléments `CLASSPATH` ou INSTANCEPATH).

En cas d'échec, un code d'erreur est renvoyé par le serveur. Les codes d'erreurs possibles sont :

- CIM_ERR_ACCESS_DENIED : l'accès à l'information n'est pas autorisé ;
- CIM_ERR_NOT_SUPPORTED : l'opération n'est pas implantée par le serveur ;
- CIM_ERR_INVALID_NAMESPACE : l'espace de nommage cible n'existe pas ;
- CIM_ERR_INVALID_PARAMETER : l'un des paramètres de l'opération est mal défini ;
- CIM_ERR_FAILED : l'opération n'a pas pu aboutir pour une raison non spécifiée.

6.6.2 L'opération AssociatorNames

L'opération **AssociatorNames** est une version allégée de l'opération **Associators**. Elle permet à un client de demander à un serveur de sélectionner un ensemble d'objets CIM (classes ou instances) selon les mêmes critères que ceux de l'opération **Associators**, mais de ne lui renvoyer que les noms des objets sélectionnés. Les paramètres de cette opération sont donnés dans la figure 40.

```
1 <objectPath>*  AssociatorNames (
2     [IN] <objectName>  ObjectName,
3     [IN,OPTIONAL,NULL] <className>  AssocClass = NULL,
4     [IN,OPTIONAL,NULL] <className>  ResultClass = NULL,
5     [IN,OPTIONAL,NULL] string  Role = NULL,
6     [IN,OPTIONAL,NULL] string  ResultRole = NULL
7 )
```

FIG. 40 – *L'opération AssociatorNames*

Les paramètres de l'opération sont identiques à ceux définis pour l'opération `Associators`.

En cas de succès, l'opération retourne la liste des noms d'objets sélectionnés. Le type de retour `ObjectPath` correspond à l'un des éléments XML `CLASSPATH` ou `INSTANCEPATH` (voir sec. 7.1) permettant de représenter en XML les nom d'une classe ou une instance CIM.

En cas d'échec un code d'erreur est renvoyé par le serveur. Les codes d'erreurs possibles sont identiques à ceux de l'opération **Associators**.

6.6.3 L'opération References

L'opération **References** permet à un client de demander la liste des associations (instances ou définitions de classes) référencant un objet CIM donné (instance ou classe). Les paramètres de cette opération sont donnés dans la figure 41.

Les paramètres `ObjectName`, `ResultClass`, `Role`, `IncludeClassOrigin` sont identiques à ceux définis dans l'opération **Associators**.

Le paramètre `PropertyList` est optionnel et constitue un filtre pour l'opération. Il permet de ne retourner dans les réponses possibles que les définitions des attributs désignés dans la liste.

```
1 <objectWithPath>*  References (
2     [IN] <objectName>  ObjectName,
3     [IN,OPTIONAL,NULL] <className>  ResultClass = NULL,
4     [IN,OPTIONAL,NULL] string  Role = NULL,
5     [IN,OPTIONAL] boolean  IncludeQualifiers = false,
6     [IN,OPTIONAL] boolean  IncludeClassOrigin = false,
7     [IN,OPTIONAL,NULL] string  PropertyList[] = NULL
8 )
```

FIG. 41 – *L'opération References*

En cas de succès, l'opération retourne la liste des associations correspondantes à la requête. En cas d'échec, Le serveur retourne un code d'erreur. Les codes d'erreurs possibles sont identiques à ceux de l'opération **Associators**.

6.6.4 L'opération ReferenceNames

L'opération **ReferenceNames** est une version allégée de l'opération **References**. Elle permet à un client de demander à un serveur de sélectionner une liste d'objets associations selon les mêmes critères que ceux définis pour l'opération **References**. Elle ne lui renvoie que les noms des objets sélectionnés. Les paramètres de cette opération sont donnés dans la figure 42.

```
1 <objectPath>*  ReferenceNames (
2     [IN] <objectName>  ObjectName,
3     [IN,OPTIONAL,NULL] <className>  ResultClass = NULL,
4     [IN,OPTIONAL,NULL] string  Role = NULL
5 )
```

FIG. 42 – *L'opération ReferenceNames*

Les paramètres de l'opération sont identiques à ceux définis pour l'opération **References**.

En cas de succès l'opération retourne la liste des noms des objets correspondants à la requête. En cas d'échec, le serveur retourne un code d'erreur. Les codes d'erreurs possibles sont identiques à ceux de l'opération **Associators**.

6.7 Les opérations d'exécution de requête

Ce profil ne comporte qu'une unique opération qui est une demande d'exécution d'une requête.

Au niveau de l'implantation, ce profil ne dépend que des opérations de lecture de base.

6.7.1 L'opération ExecQuery

Unique opération du profil, elle permet de lancer une requête sur les objets maintenus dans l'espace de nommage cible. Les paramètres de cette opération sont donnés dans la figure 43.

```
1 <object>*  ExecQuery (
2     [IN] string  QueryLangage,
3     [IN] string  Query
4 )
```

FIG. 43 – *L'opération ExecQuery*

Le paramètre **QueryLangage** (ligne 2) permet de désigner le langage dans lequel la requête est spécifiée. Le seul langage de requêtes disponible à ce jour est le *WBEM Query Langage* (WQL) [11] récemment proposé

par le DMTF. En réalité, ce n'est qu'un sous-ensemble du langage SQL utilisé pour les seules opérations de lectures de données.

Le paramètre **Query** (ligne 3) permet au client de donner la spécification de la requête qu'il désire exécuter sur l'espace de nommage cible maintenu par le serveur.

En cas de succès, l'opération retourne la liste des objets sélectionnés par la requête. Le type de retour `object` correspond à l'un des éléments XML `VALUE.OBJECT`, `VALUE.OBJECTWITHLOCALPATH` ou `VALUE.OBJECTWITHPATH` (voir sec. 7.1). Ces éléments que nous ne détaillons pas ici, permettent de donner la représentation XML de l'objet, et selon le cas joindre la représentation XML de l'identificateur local ou global de l'objet.

En cas d'échec, le serveur retourne un code d'erreur. Les codes d'erreurs possibles sont les suivants :

- CIM_ERR_ACCESS_DENIED : l'accès à l'information n'est pas autorisé ;
- CIM_ERR_NOT_SUPPORTED : l'opération n'est pas implantée par le serveur ;
- CIM_ERR_INVALID_NAMESPACE : l'espace de nommage cible n'existe pas ;
- CIM_ERR_INVALID_PARAMETER : l'un des paramètres de l'opération est mal défini ;
- CIM_ERR_QUERY_LANGAGE_NOT_SUPPORTED : le langage de la requête n'est pas reconnu par le serveur ;
- CIM_ERR_INVALID_QUERY : la requête spécifiée est incorrecte ;
- CIM_ERR_FAILED : l'opération n'a pas pu aboutir pour une raison non spécifiée.

6.8 Les opérations de déclaration de qualifieur

Ce profil fonctionnel regroupe toutes les opérations de lecture, création, suppression ou modification de la définition d'un qualifieur.

Au niveau de l'implantation, ce profil nécessite le support des opérations de manipulation de schéma.

6.8.1 L'opération GetQualifier

L'opération **GetQualifier** permet à un client de demander à un serveur la définition d'un qualifieur donné dans l'espace de nommage cible. La définition de l'opération est donnée dans la figure 44.

```
1 <qualifierDecl>*  GetQualifier (
2     [IN] string  QualifierName
3 )
```

FIG. 44 – *L'opération GetQualifier*

L'unique paramètre de l'opération `QualifierName` (ligne 2) permet de désigner le nom du qualifieur dont la définition est demandée par le client.

En cas de succès, l'opération retourne la définition du qualifieur correspondant à la requête. Le type de retour `qualifierDecl` correspond à l'élément `QUALIFIER.DECLARATION` (voir sec. 7.1) permettant de donner une représentation XML d'une définition de qualifieur.

En cas d'échec, le serveur renvoie un code d'erreur. Les codes d'erreurs possibles sont les suivants :

- CIM_ERR_ACCESS_DENIED : l'accès à l'information n'est pas autorisé ;
- CIM_ERR_NOT_SUPPORTED : l'opération n'est pas implantée par le serveur ;
- CIM_ERR_INVALID_NAMESPACE : l'espace de nommage cible n'existe pas ;
- CIM_ERR_INVALID_PARAMETER : l'un des paramètres de l'opération est mal défini ;
- CIM_ERR_NOT_FOUND : la définition du qualifieur spécifié n'existe pas ;
- CIM_ERR_FAILED : l'opération n'a pas pu aboutir pour une raison non spécifiée.

6.8.2 L'opération SetQualifier

L'opération **SetQualifier** permet de créer ou de modifier une définition de qualifieur dans l'espace de nommage cible. La définition de l'opération est donnée dans la figure 45.

L'unique paramètre de l'opération `QualifierDeclaration` (ligne 2) permet de donner la définition du qualifieur à rajouter ou à modifier dans l'espace de nommage cible.

```
1 void  SetQualifier (
2     [IN] <qualifierDecl>  QualifierDeclaration
3 )
```

FIG. 45 – *L'opération SetQualifier*

En cas de succès, aucune donnée n'est retournée. En cas d'échec, le serveur renvoie un code d'erreur. Les codes d'erreurs possibles sont :

- CIM_ERR_ACCESS_DENIED : l'accès à l'information n'est pas autorisé ;
- CIM_ERR_NOT_SUPPORTED : l'opération n'est pas implantée par le serveur ;
- CIM_ERR_INVALID_NAMESPACE : l'espace de nommage cible n'existe pas ;
- CIM_ERR_INVALID_PARAMETER : l'un des paramètres de l'opération est mal défini ;
- CIM_ERR_FAILED : l'opération n'a pas pu aboutir pour une raison non spécifiée.

6.8.3 L'opération DeleteQualifier

L'opération **DeleteQualifier** permet à un client de demander à un serveur la suppression d'une définition de qualifieur donné dans l'espace de nommage cible. La définition de l'opération est donnée dans la figure 46.

```
1 void  DeleteQualifier (
2     [IN] string  QualifierName
3 )
```

FIG. 46 – *L'opération DeleteQualifier*

L'unique paramètre de l'opération `QualifierName` (ligne 2) permet de désigner le nom du qualifieur à supprimer de l'espace de nommage cible.

En cas de succès, l'opération ne retourne aucune donnée. En cas d'échec, le serveur renvoie un message d'erreur. Les codes d'erreurs possibles sont identiques à ceux de l'opération **GetQualifier**.

6.8.4 L'opération EnumerateQualifiers

L'opération **EnumerateQualifiers** permet à un client de demander à un serveur de lui renvoyer la liste des définitions des qualifieurs spécifiés dans l'espace de nommage cible.

La définition de l'opération est donnée dans la figure 47. Elle ne comprend aucun paramètre et retourne en cas de succès les définitions de qualifieurs définis dans l'espace de nommage cible. En cas d'echec, le serveur renvoie un code d'erreur. Les codes d'erreurs possibles sont identiques à ceux de l'opération **SetQualifier**.

```
1 <qualifierDecl>*  EnumerateQualifiers (
2 )
```

FIG. 47 – *L'opération EnumerateQualifiers*

6.9 Résumé

Les opérations CIM telles qu'elles sont actuellement définies par le DMTF correspondent dans leur totalité à une interaction du type client/serveur. Elles permettent toutes sortes de manipulations de lecture et d'écriture à deux niveaux distincts : accès aux donnés (objets gérés et relations) et accès aux modèles (classes, associations et qualifieurs).

Cependant, aucune opération n'a été définie pour des interactions du type producteur/consommateur, ce qui, à notre avis, présente une lacune de taille que le DMTF se doit de résoudre dans les prochaines versions du standard.

De plus, certaines de ces opérations, et surtout les opérations de manipulation du modèle (classes, associations et définitions de qualifieurs) sont encore mal spécifiées. En effet, les spécifications actuelles manquent de précisions sur le comportement d'un serveur devant une demande de modification ou de suppression d'une définition de classe vis-à-vis des instances et des sous-classes de celle-ci. Le cas des opérations de modification ou de suppression d'une définition de qualifieur est encore plus frappant, et pour lesquels aucune information n'est donné dans la norme sur l'impact qu'elles doivent avoir sur les instances de qualifieurs "éparpillées" sur toute la base de gestion.

7 Le protocole de communication XML/HTTP

Le protocole de communication proposé par le DMTF s'appuye sur deux standards du Web : XML (eXtensible Markup Langage) [26] pour l'encodage et HTTP (HyperText Transfer Protocol) [15] pour le transport. La figure 48, schématisant une interaction client-serveur entre deux entités WBEM, illusre le couplage XML/HTTP proposé par le DMTF pour le transport de l'information CIM.

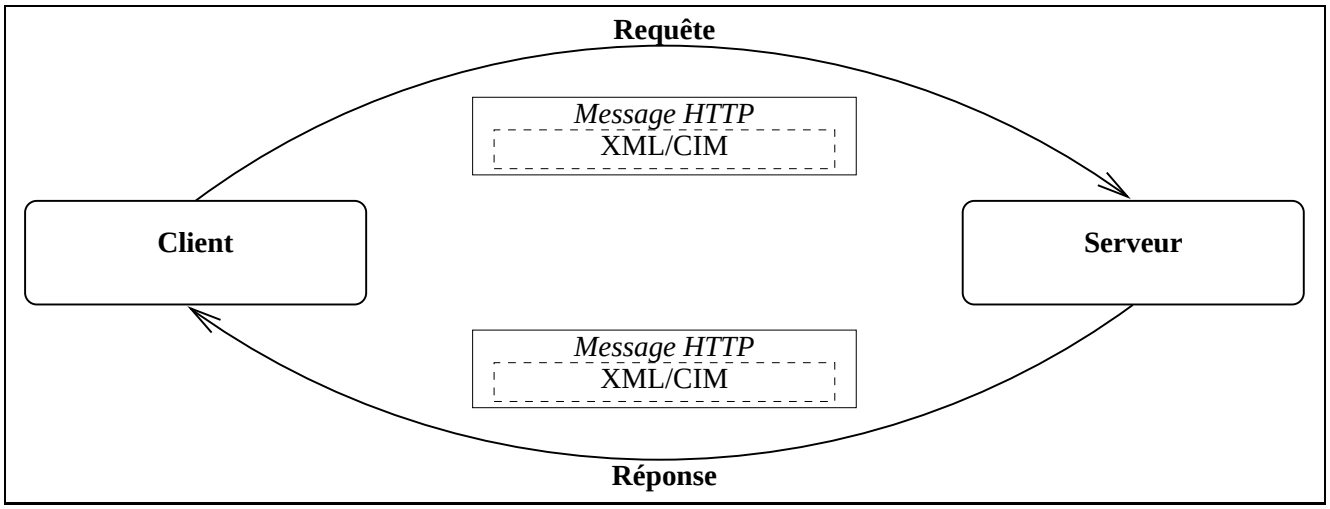


FIG. 48 – *La communication de l'information dans WBEM*

Dans sa version actuelle, la représentation XML donnée par le DMTF [10] permet d'encoder les spécifications de la vesion 2.2 du modèle CIM, ainsi que la version 1.0 des opérations CIM.

L'objet de cette section n'est pas de présenter le standard dans sa totalité, mais d'en donner les éléments de base afin de permettre la compréhension du modèle de communication dans WBEM.

7.1 La représentation XML

Le langage XML, sous-ensemble de SGML (Standard Generalized Markup Langage) [20], offre un moyen puissant et extensible pour une représentation "conviviale"[18] sous format texte de tout ensemble structuré de données. Cette représentation consiste à encapsuler les données dans les entités de base XML : les **éléments**.

La structure d'un document XML, i.e. les **éléments** utilisés et leur arborescence, est définie dans un document de grammaire appelé document DTD (Document Type Definition). La validité d'un document XML impose alors une conformité à la fois aux règles syntaxiques de XML ainsi qu'aux règles sémantiques données dans le document de grammaire. Pour plus d'informations sur XML et sur son intérêt pour la representation de l'information de gestion CIM, nous renvoyons le lecteur à [26] et [6].

Comme pour l'intégration des modèles de l'information dans CIM (voir sec. 8), il existe deux différentes stratégies pour la représentation en XML de l'information CIM :

- la stratégie de représentation du méta-modèle, comparable à la traduction de *technique*, consiste à définir un document DTD spécifiant les éléments XML correspondants à chacun des composants du méta-modèle CIM (classe, propriété, méthode,...) ;

18. L'expression anglaise *Human Readable* est certainement plus parlante !

– la stratégie de représentation de modèle, comparable à la traduction de *recast*, consiste à définir un document DTD spécifiant les éléments XML correspondants à chaqun des composants du modèle (les classes et les associations du modèle CIM à représenter).

Le DMTF a opté pour la première stratégie de représentation, i.e. la représentation du méta-modèle, pour deux raisons principales :

– Un unique document DTD permet de traduire toute modélisation CIM, contrairement à la représentation de modèle qui nécessite la définition d'un nouveau document DTD pour chaque nouveau modèle. La représentation XML du méta-modèle offre ainsi un moyen standardisé et homogène pour la représentation en XML ;
– La traduction de méta-modèle permet de représenter à la fois les classes et les instances dans un document XML, contrairement à la traduction de modèle ne permettant de représenter que les instances, les classes étant traduites par des éléments XML définis dans des documents DTD.

Conformément à ce choix de représentation XML, le DMTF a défini un document DTD [7] spécifiant les éléments XML qui permettent de représenter l'information CIM à véhiculer entre deux entités WBEM. Dans cette section nous présentons les éléments les plus significatifs de cette représentation.

7.2 Représentation d'une classe CIM

La représentation d'une classe CIM en XML est donnée dans la figure 49. Elle sert à la fois pour la représentation d'une classe ainsi que pour la représentation d'une association.

```
1 <!ELEMENT CLASS
2     ( QUALIFIER * ,
3     ( PROPERTY | PROPERTY.ARRAY | PROPERTY.REFERENCE ) * ,
4     METHOD * )
5 >
6 <!ATTLIST CLASS
7     NAME        CDATA        #REQUIRED
8     SUPERCLASS  CDATA        #IMPLIED
9 >
```

FIG. 49 – *La représentation d'une classe CIM en XML*

L'élément **CLASS** possède deux attributs `NAME` et `SUPERCLASS` (lignes 6 à 9) permettant de donner le nom de la classe ainsi que le nom de la super-classe dans la représentation XML.

L'élément **QUALIFIER** (ligne 2) permet de représenter un qualifieur associé à la classe CIM.

Les éléments **PROPERTY** et **PROPERTY.ARRAY** (ligne 3) permettent de représenter une définition d'attribut de la classe CIM, le premier étant réservé aux attributs de type simple et le second aux attributs de type vecteur. La spécification de l'élement **PROPERTY** est donnée dans la figure 50.

L'élément **PROPERTY.REFERENCE** (ligne 3) est une troisième variante de l'élément **PROPERTY**, et permet de représenter une définition de référence d'une association CIM.

L'élément **METHOD** (ligne 4) permet de représenter une définition de méthode de la classe CIM. La spécification de l'élément est donnée dans la figure 51.

7.2.1 Représentation d'une propriété CIM

L'élément **PROPERTY** (figure 50) est utilisé pour représenter une propriété de type simple. Il est utilisé à la fois pour la représentation d'une définition de propriété lors de la représentation d'une classe CIM, et pour la représentation d'une initialisation de propriété lors de la représentation d'une instance CIM.

L'élément peut contenir les 2 sous-éléments et 4 attributs suivants :

– L'élément **QUALIFIER** permettant d'inclure dans la représentation XML les qualifieurs associés à la propriété.

```
<!ELEMENT PROPERTY
( QUALIFIER *,
VALUE ? )
>
<!ATTLIST PROPERTY
    NAME        CDATA                                          #REQUIRED
    TYPE        ( boolean | string | char16 | datetime |
                  sint8   | sint16 | sint32 | sint64   |
                  uint8   | uint16 | uint32 | uint64   |
                   real32  | real64 )                         #REQUIRED
    CLASSORIGIN CDATA                                          #IMPLIED
    PROPAGATED  ( true | false )                               #IMPLIED
>
```

FIG. 50 – *Représentation XML d'une propriété CIM*

- L'élément **VALUE** permettant d'associer une valeur d'initialisation à la propriété (valeur d'initialisation par défaut pour une représentation de classe, et valeur d'initialisation pour une représentation d'instance) ;
- L'attribut `NAME` : donne le nom de la propriété ;
- L'attribut `TYPE` : donne le type de la propriété ;
- L'attribut `CLASSORIGIN` : donne le nom de la classe définissant pour la première fois la propriété CIM. Il permet ainsi d'alléger les applications clientes en résolvant rapidement l'indéterminisme apparent dû au mécanisme d'héritage des propriétés (indéterminisme apparent dans le sens ou l'héritage masque l'origine des propriétés des objets, i.e. la classe définissant pour la première fois la propriété) ;
- L'attribut `PROPAGATED` : indique que la représentation de la propriété a été inclue dans la représentation de la classe (ou de l'instance) par héritage, ou par spécification explicite.

Le DMTF a défini deux autres variantes de l'élément **PROPERTY** : l'élément **PROPERTY.ARRAY** permettant de représenter une propriété de type vecteur, et l'élément **PROPERTY.REFRENCE** permettent de représenter une référence dans une association. Les trois variantes diffèrent les unes des autres par les sous-éléments de représentation des valeurs des propriétés (initialisation dans une instance, ou valeur par défaut dans une définition de class) : **VALUE** pour une valeur de type simple, **VALUE.ARRAY** pour une valeur de type vecteur et **VALUE.REFERENCE** pour une valeur de type référence vers un objet (classe ou instance).

7.2.2 Représentation d'une méthode CIM

L'élément **METHOD** est utilisé pour représenter une définition de méthode CIM. La spécification XML de cet élément est donnée dans la figure 51.

L'élément **METHOD** peut contenir 5 types de sous-éléments :

- l'élément **QUALIFIER** permet de représenter chacun des qualifieurs associés à la méthode ;
- l'élément **PARAMETER** permet de représenter un paramètre de type simple ;
- l'élément **PARAMETER.ARRAY**, variante de l'élément **PARAMETER**, permet de représenter un paramètre de type vecteur ;
- l'élément **PARAMETER.REFERENCE**, variante de l'élément **PARAMETER**, permet de représenter un paramètre de type référence simple ;
- l'élément **PARAMETER.REFARRAY**, variante de l'élément **PARAMETER**, permet de représenter un paramètre de type vecteur de références.

Tout comme l'élément **PROPERTY**, l'élément **METHOD** possède 4 attributs permettant de donner le nom de la méthode, son type (type de retour), sa classe d'origine et sa caractéristique de propagation (attribut `PROPAGATED`).

```
<!ELEMENT METHOD
( QUALIFIER *,
( PARAMETER | PARAMETER.REFERENCE |
PARAMETER.ARRAY | PARAMETER.REFARRAY ) *
>
<!ATTLIST METHOD
    NAME        CDATA                                          #REQUIRED
    TYPE        ( boolean | string | char16 | datetime |
                  sint8   | sint16 | sint32 | sint64   |
                  uint8   | uint16 | uint32 | uint64   |
                  real32  | real64 )                         #IMPLIED
    CLASSORIGIN CDATA                                          #IMPLIED
    PROPAGATED  ( true | false )                               #IMPLIED
>
```

FIG. 51 – *Représentation XML d'une méthode CIM*

7.2.3 Représentation d'un qualifieur

L'élément **QUALIFIER** (figure 52) est utilisé pour représenter une instance de qualifieur associé à un élément CIM (classe, instance, propriété, méthode et paramètre).

```
<!ELEMENT QUALIFIER
( VALUE |
VALUE.ARRAY )
>
<!ATTLIST QUALIFIER
    NAME         CDATA                                         #REQUIRED
    TYPE         ( boolean | string | char16 | datetime |
                   sint8   | sint16 | sint32 | sint64   |
                   uint8   | uint16 | uint32 | uint64   |
                   real32  | real64 )                        #REQUIRED
    PROPAGATED   ( true | false )                              #IMPLIED
    OVERRIDABLE  ( true | false )                              'true'
    TOSUBCLASS   ( true | false )                              'true'
    TOINSTANCE   ( true | false )                              'false'
    TRANSLATABLE ( true | false )                              'false'
>
```

FIG. 52 – *Représentation XML d'un qualifieur CIM*

L'élément **QUALIFIER** comporte 2 sous-éléments et 4 attributs XML :

- L'élément **VALUE** utilisé pour représenter une valeur associée à un qualifieur de type simple ;
- L'élément **VALUE.ARRAY** utilisé pour représenter les valeurs associées à un qualifieur de type vecteur ;
- L'attribut `NAME` utilisé pour donner le nom du qualifieur ;
- L'attribut `TYPE` utilisé pour donner le type du qualifieur ;
- L'attribut `PROPAGATED` indiquant si la définition du qualifieur est explicite, ou héritée ;
- Quatre attributs définissant la règle de propagation associée au qualifieur (**Flavor**, voir sec. 4.2) :
  l'attribut `OVERRIDABLE` correspond à la règle de propagation `ENABLE_OVERRIDE` de la spécification MOF d'un qualifieur. S'il est donné avec la valeur `true`, l'attribut indique que le qualifieur peut être instancié diffèremment dans une sous-classe ;

l'attribut TOSUBCLASS correspond à la règle de propagation TO_SUBCLASS de la spécification MOF d'un qualifieur. S'il est donné avec la valeur true, l'attribut indique que l'instance du qualifieur doit être propagée par héritage ;

l'attribut TOINSTANCE indique la règle de propagation du qualifieur vis-à-vis de l'instanciation[19]. S'il est donné avec la valeur true, le qualifieur doit être propagé aux instances de la classe à laquelle il est associé ;

l'attribut TRANSLATABLE correspond à la règle de propagation TRANSLATABLE de la spécification MOF d'un qualifieur. S'il est donné avec la valeur true, l'attribut indique que le qualifieur peut être "cloné" dans d'autres langues.

7.2.4 Un exemple de représentation d'une classe

L'exemple donné ici (figure 53) est extrait de la représentation de la classe CIM_ComputerSystem du schéma de base CIM (*Core Schema*). La spécification MOF de cette classe est donnée dans la figure 9 (section 4.1).

```
1  < CLASS NAME="CIM_ComputerSystem" SUPERCLASS="CIM_System" >

2    < QUALIFIER NAME="Description" TYPE="string" TOSUBCLASS="false" >
3      < VALUE > Computer System logical model < /VALUE >
4    < /QUALIFIER >

5    < PROPERTY NAME="Name" TYPE="string"
6          CLASSORIGIN="CIM_System" PROPAGATED="true" >
7      < QUALIFIER NAME="Key" TYPE="boolean" PROPAGATED="true" >
8        < VALUE > TRUE < /VALUE >
9      < /QUALIFIER >
10   < /PROPERTY >

11   < PROPERTY NAME="NameFormat" TYPE="string"
12         CLASSORIGIN="CIM_System" PROPAGATED="false" >
13     < QUALIFIER NAME="Override" TYPE="string" TOSUBCLASS="false" >
14       < VALUE> "NameFormat" < /VALUE >
15     < /QUALIFIER >
16   < /PROPERTY >

17   < PROPERTY NAME="LocalId" TYPE="uint32"
18         CLASSORIGIN="CIM_ComputerSystem" PROPAGATED="false" >
19   < /PROPERTY >

20   < METHOD NAME="Reboot" TYPE="boolean" >
21     < PARAMETER NAME="Time" TYPE="datetime" >
22     < /PARAMETER >
23   < /METHOD >
24 < /CLASS >
```

FIG. 53 – *Exemple de représentation d'une classe CIM en XML*

L'exemple donné n'est pas complet mais permet d'illustrer la représentation XML des différents éléments de définition d'une classe CIM.

L'élément **CLASS**, élément racine de la représentation, donne le nom de la classe ainsi que celui de sa super-classe (ligne 1).

L'élément **Qualifier** (lignes 2 à 4), inclu dans l'élément **CLASS**, permet de donner la représention XML du qualifieur Description associé à la classe. Il en donne le nom (attribut NAME), le type (attribut TYPE), la règle de propagation (attribut TOSUBCLASS) et la valeur (sous-élément **VALUE**).

19. Cette règle de propagation n'est pas définie dans la version actuelle de CIM. Elle sera peut être rajoutée dans une version ultérieure.

Ensuite, nous retrouvons les représentations de chacune des propriétés de la classe données par les éléments **PROPERTY**.

La première propriété (lignes 5 à 9) représente la propriété `Name` héritée de la classe `CIM_System`. Ceci est spécifiée par l'attribut `CLASSORIGIN`. L'attribut `PROPAGATED` fixé à `true` précise que la propriété a été héritée sans surcharge (i.e. sans re-définition). L'élément possède un sous-élément **QUALIFIER** représentant le qualifieur `Key` associé à la propriété.

La deuxième propriété (lignes 10 à 14) représente la propriété `NameFormat` héritée de la classe `CIM_System` et surchargée dans la classe `CIM_ComputerSystem`. Ceci est spécifié grâce aux attributs `CLASSORIGIN` et `PROPAGATED`. L'élément possède un sous-élément **QUALIFIER** représentant le qualifieur `Override` associé à la propriété.

La troisième propriété (lignes 15 à 16) représente la propriété `LocalId` définie par la classe.

L'élément **METHOD** (lignes 17 à 20) donne une représentation XML de la méthode `Reboot` définie par la classe. Le type de retour de la méthode est donné par l'attribut `TYPE` (ligne 17) associé à l'élément. L'unique paramètre `Time` défini par la méthode est représenté par le sous-élément **PARAMETER** (lignes 18 et 19) donnant le nom et le type du paramètre.

7.3 Représentation d'une instance

La représentation d'une instance CIM en XML est donnée dans la figure 54. Elle sert à la fois pour la représentation d'une instance d'objet géré ainsi que pour la représentation d'une instance de relation.

```
1 <!ELEMENT INSTANCE
2 ( QUALIFIER * ,
3 ( PROPERTY| PROPERTY.ARRAY| PROPERTY.REFERENCE ) * )
4 >
5 <!ATTLIST INSTANCE
6     CLASSNAME         CDATA         #REQUIRED
7 >
```

FIG. 54 – *L'encodage d'une instance CIM en XML*

L'élément **INSTANCE** n'a qu'un seul attribut `CLASSNAME` qui permet de donner le nom de la classe dont l'instance est issue.

L'élément **QUALIFIER** permet de représenter un qualifieur associé à l'instance CIM.

Les éléments **PROPERTY** et **PROPERTY.ARRAY** permettent de représenter les propriétés de l'instance. Le premier étant réservé aux propriétés de type simple, et le second aux propriétés de type vecteur.

L'élément **PROPERTY.REFERENCE** permet de représenter les valeurs des références pour une instance d'association CIM.

Pour illustrer la représentation XML des instances CIM, nous donnons dans la figure 55 la représentation XML d'une instance de la classe `CIM_ComputerSystem`. La spécification MOF de l'instance représentée ici est donnée dans la section 4.4 (voir figure 19).

L'élément **INSTANCE** donne une représentation de la définition de l'instance. Dans cette représentation, on retrouve le nom de la classe dont l'instance est issue (attribut `CLASSNAME`), ainsi que les représentations de chacune des initialisations de propriétés de l'instance (éléments **PROPERTY**).

7.4 Représentation des identificateurs d'objets CIM

Certains éléments XML définis par le DMTF, tels que `VALUE.OBJECTWITHPATH`, `VALUE.OBJECTWITHLOCALPATH` ou `VALUE.NAMEDINSTANCE`, que nous ne détaillerons pas ici [20], permettent de donner la représentation XML d'un objet (l'élément `CLASS` pour les classes et l'élément `INSTANCE` pour les instances) précédée de la représentation XML du nom de l'objet `ObjectName` défini dans le nommage CIM (voir sec. 3.2).

Les représentations des noms d'objets diffèrent selon la nature de l'objet (classe ou instance), ainsi que selon le type de nommage (local ou global). Dans cette section nous détaillons la représentation recommandée par

20. Pour plus d'informations sur ces éléments nous renvoyons le lecteur à [6]

```
< INSTANCE CLASSNAME ="CIM_ComputerSystem" >

  < PROPERTY NAME="Name" TYPE="string" >
    < VALUE > 152.81.11.87 < /VALUE >
  < /PROPERTY

  < PROPERTY NAME="NameFormat" TYPE="string" >
    < VALUE > IP < /VALUE >
  < /PROPERTY

  < PROPERTY NAME="LocalId" TYPE="uint32" >
    < VALUE > 12 < /VALUE >
   < /PROPERTY

< /INSTANCE >
```

FIG. 55 – *Exemple de représentation d'une instance CIM en XML*

le DMTF pour le nommage des instances et nous renvoyons le lecteur à [6] pour plus d'informations sur le nommage des classes[21].

7.4.1 Représentation d'une référence vers un espace de nommage

Le nom d'un espace de nommage est représenté à l'aide de l'élément **NAMESPACEPATH** qui correspond au champs `NamespaceHandle` dans le nommage CIM (voir sec. 3.2). La spécification de cet élément accompagnée d'un exemple de représentation sont donnés dans la figue 56.

Définition de l'élément

```
<!ELEMENT  NAMESPACEPATH
  ( HOST , LOCALNAMESPACEPATH )
>

<!ELEMENT  HOST (#PCDATA) >

<!ELEMENT  LOCALNAMESPACEPATH ( NAMESPACE + ) >

<!ELEMENT  NAMESPACE EMPTY >
<!ATTLIST  NAMESPACE
    NAME    CDATA    #REQUIRED >
```

Exemple de représentation

```
< NAMESPACEPATH >
  < HOST > dolcourt.loria.fr < /HOST >
  < LOCALNAMESPACEPATH >
    < NAMESPACE NAME="root"  / >
    < NAMESPACE NAME="cimv2"  / >
  < /LOCALNAMESPACEPATH>
< /NAMESPACEPATH>
```

FIG. 56 – *L'élément* **NAMESPACEPATH**

L'élément **NAMESPACEPATH** se compose de deux sous-éléments :

– l'élément **HOST** permettant de donner le nom du serveur hôte ;

21. Les deux représentations sont équivalentes sur le principe, et ne diffèrenet que sur les compositions des éléments.

– l'élément **LOCALNAMESPACEPATH** permettant de donner le nom de l'espace de nommage, i.e. la liste des noms d'espaces de nommages à parcourir jusqu'à l'espace de nommage ciblé.

Cette spécification est illustrée sur un exemple (lignes 9 à 15) donnant la représentation XML d'une référence à l'espace de nommage `/root/cimv2` hébergé par le serveur `dolcourt.loria.fr`.

7.4.2 Représentation d'une référence vers une instance CIM

La référence vers une instance est représentée à l'aide de l'élément **INSTANCENAME** qui correspond au champs `ModelPath` dans le nommage des instances CIM (voir sec. 3.2). La spécification de cet élément est donnée dans la figure 57.

```
1  <!ELEMENT INSTANCENAME
2    ( KEYBINDING * | KEYVALUE ? | VALUE.REFERENCE ? )
3  >
4  <!ATTLIST INSTANCENAME
5      CLASSNAME    CDATA    #REQUIRED
6  >

7  <!ELEMENT KEYBINDING
8    ( KEYVALUE | VALUE.REFERENCE )
9  >
10 <!ATTLIST KEYBINDING
11     NAME    CDATA    #REQUIRED
12 >

13 <!ELEMENT KEYVALUE (#PCDATA) >
14 <!ATTLIST KEYVALUE
15     VALUETYPE    ( string | boolean | numeric )    'string'
16 >
```

FIG. 57 – *Définition de l'élément* **INSTANCENAME**

L'élément **INSTANCENAME** comprend un unique attribut `CLASSNAME` permettant de donner le nom de la classe dont l'instance est issue.

Les sous-éléments possibles pour cet élément sont au nombre de trois. L'utilisation de l'un ou de l'autre de ces sous-éléments dépend de la nature de l'instance (classe ou association) ainsi que du nombre de ses attributs clefs :

– l'élément **KEYBINDING** (voir lignes 7 à 12) permet de donner le nom et la valeur d'un attribut clef pour les instances de classes spécifiants plusieurs attributs clefs ;
– l'élément **KEYVALUE** (voir lignes 13 à 16) permet de donner la valeur et le type d'un attribut clef ;
– l'élément **VALUE.REFERENCE** permet de donner la valeur prise par une référence spécifiée comme clef pour le nommage des instances d'une association.

Quelques exemples illustrants les représentations des références vers des instances sont donnés dans la figure 58.

Le premier exemple donne la représentation d'une référence vers une instance avec une propriété clef unique.

Le second exemple donne la représentation d'une référence vers une instance d'association avec un unique attribut clef du type référence vers un objet.

Le dernier exemple donne la représentation d'une référence vers une instance possédant deux attributs clefs.

7.5 Représentation d'une définition de qualifieur

La représentation d'une définition de qualifieur CIM en XML est donnée dans la figure 59.

L'élément **QUALIFIER.DECLARATION** contient trois sous-éléments et cinq attributs :

– L'élément **SCOPE** permet de donner les entités CIM cibles du qualifieur ;

Exemple 1

```
< INSTANCENAME CLASSNAME="classeA" >
  < KEYVALUE VALUETYPE="numeric" > 1 < /KEYVALUE >
< /INSTANCENAME >
```

Exemple 2

```
< INSTANCENAME CLASSNAME="assosiationA" >
  < VALUE.REFERENCE>
    < INSTANCENAME CLASSNAME="classeA" >
      < KEYVALUE VALUETYPE="numeric" > 1 < /KEYVALUE >
    < INSTANCENAME
  < /VALUE.REFERENCE
 /INSTANCENAME
```

Exemple 3

```
< INSTANCENAME CLASSNAME="classB" >
  < KEYBINDING NAME="clefA" >
    < KEYVALUE VALUETYPE="string" > dolcourt < /KEYVALUE >
  < /KEYBINDING >
  < KEYBINDING NAME="clefB" >
    < KEYVALUE VALUETYPE="numeric" > 1 < /KEYVALUE >
  < /KEYBINDING
< /INSTANCENAME >
```

FIG. 58 – *Exemples de représentation d'un nom d'instance CIM*

```
<!ELEMENT QUALIFIER.DECLARATION
( SCOPE ?,
(VALUE | VALUE.ARRAY)? )
>
<!ATTLIST QUALIFIER.DECLARATION
    NAME         CDATA                                        #REQUIRED
    TYPE         ( boolean | string | char16 | datetime |
                   sint8   | sint16 | sint32 | sint64   |
                   uint8   | uint16 | uint32 | uint64   |
                   real32  | real64 )                        #REQUIRED
    ISARRAY      ( true | false )                             #IMPLIED
    ARRAYSIZE    CDATA                                        #IMPLIED
    OVERRIDABLE  ( true | false )                             'true'
    TOSUBCLASS   ( true | false )                             'true'
    TOINSTANCE   ( true | false )                             'false'
    TRANSLATABLE ( true | false )                             'false'
>
```

FIG. 59 – *La représentation d'une définition d'un qualifieur CIM en XML*

- L'élément **VALUE** permet de représenter une valeur d'initialisation par défaut pour un qualifieur de type simple ;
- L'élément **VALUE.ARRAY** permet de représenter les valeurs d'initialisation par défaut pour un qualifieur de type vecteur ;
- L'attribut `NAME` permet de donner le nom du qualifieur ;
- L'attribut `TYPE` permet de donner le type du qualifieur ;
- L'attribut `ISARRAY` permet d'indiquer si le qualifieur est de type simple ou de type vecteur ;
- L'attribut `ArraySize` permet de donner la taille du vecteur pour un qualifieur de type vecteur ;
- Les attributs `OVERRIDABLE`, `TOSUBCLASS`, `TOINSTANCE` et `TRANSLATABLE` permettent de donner les règles de propagation du qualifieur, i.e. le **flavor** dans la terminologie CIM.

Pour illustrer cette spécification, nous donnons deux exmples de représentations de définitions de qualifieurs dans la figure 60. Le premier exemple donne la définition du méta qualifieur `Association`, et le second celui du qualifieur `Descriptions` définis par le DMTF.

Exemple 1

```
1 < QUALIIFER.DECLARATION
2   NAME="Association" TYPE="boolean" ISARRAY="false" OVERRIDABLE="false" >
3     < SCOPE CLASS="true" ASSOCIATION="true"  / >
4     < VALUE > false < /VALUE >
5 < /QUALIFIER.DECLARATION >
```

Exemple 2

```
6 < QUALIFIER.DECLARATION
7   NAME="Description" TYPE="string" ISARRAY="false" TRANSLATABLE="true" >
8     < SCOPE CLASS="true" ASSOCIATION="true" INDICATION="true"
9       REFERENCE="true" PROPERTY="true" METHOD="true" PARAMETER="true"  / >
10 < /QUALIFIER.DECLARATION >
```

FIG. 60 – *Exemples de représentations de définition de qualifieurs*

7.6 Représentation d'une opération CIM

Les opérations CIM invoquées par un client, ainsi que les réponses retournées par un serveur[22], sont véhiculées à travers le réseau sous une représentation XML dans les unités de données du protocole HTTP. Dans cette section, nous présentons la représentation des requêtes et réponses CIM dans le langage XML tel que le recommande la version 1.0 [9] du standard proposé par le DMTF.

7.6.1 Représentation d'un message

Les unités de données HTTP échangées entre deux entités WBEM sont représentées par l'élément XML **MESSAGE**. Cet élément sert d'élément racine pour la représentation d'une invocation d'opération par le client ou de la réponse retournée par le serveur. La description de l'élément **MESSAGE** est donnée dans la figure 61 :

Le message ne peut contenir qu'un seul sous-élément parmi les 4 suivants :

- L'élément **SIMPLEREQ** donne une représentation d'une requête simple (lignes 10 à 11) : il permet d'invoquer une opération CIM (méthode intrinsèque), ou de demander l'exécution d'une méthode sur un objet (méthode extrinsèque) ;
- L'élément **MULTIREQ** donne une représentation d'une requête multiple : il permet au client de composer plusieurs requêtes simples et de les envoyer dans un même message ;
- L'élément **SIMPLERSP** donne une représentation d'une réponse simple (lignes 13 à 15) : il permet au serveur de répondre à une requête simple ;
- L'élément **MULTIRSP** donne une representation d'une réponse multiple : il permet au serveur de composer plusieurs reponses suite à un appel d'une requête multiple.

22. Rappelons que dans la version actuelle de la norme, les opérations CIM sont limitées aux interactions du type client/serveur.

```
1  <!ELEMENT MESSAGE
2      ( SIMPLEREQ | MULTIREQ
3      | SIMPLERSP | MULTIRSP )
4  >
5  <!ATTLIST MESSAGE
6      ID               CDATA    #REQUIRED
7      PROTOCOLVERSION  CDATA    #REQUIRED
8  >

9  <!ELEMENT SIMPLEREQ
10     ( IMETHODCALL | METHODCALL )
11 >
12 <!ELEMENT SIMPLERSP
13     ( IMETHODRESPONSE | METHODRESPONSE )
14 >

15 <!ELEMENT MULTIREQ
16     ( SIMPLEREQ , SIMPLEREQ + )
17 >
18 <!ELEMENT MULTIRSP
19     ( SIMPLERSP, SIMPLERSP + )
20 >
```

FIG. 61 – *Représentation d'un message CIM en XML*

7.6.2 Représentation d'un appel de méthode

Pour représenter un appel de méthode, le DMTF a défini deux éléments **IMETHODCALL** et **METHODCALL** correspondant chacun à une famille de méthodes : méthodes intrinsèques ayant pour cibles des espaces de nommages et méthodes extrinsèques ayant pour cibles des classes ou des instances CIM.

Mais d'une manière générale, la représentation permet de spécifier le nom de la méthode invoquée, et de spécifier la cible de la méthode ainsi que la liste de ses paramètres (voir fig. 62).

Appel d'une méthode instrinsèque

```
1  <!ELEMENT IMETHODCALL
2      ( LOCALNAMESPACEPATH , IPARAMVALUE * )
3  >
4  <!ATTLIST IMETHODCALL
5      NAME    CDATA    #REQUIRED
6  >
```

Appel d'une méthode extrinsèque

```
7  <!ELEMENT METHODCALL
8      ( ( LOCALINSTANCEPATH | LOCALCLASSPATH ) , PARAMVALUE * )
9  >
10 <!ATTLIST METHODCALL
11     NAME    CDATA    #REQUIRED
12 >
```

FIG. 62 – *Représentation d'un appel de méthode*

L'élément **IMETHODCALL** (lignes 1 à 6) permet d'appeler une méthode intrinsèque. Il permet de donner le nom de la méthode (l'attribut `NAME`, ligne 5), une référence vers l'espace de nommage cible de l'opération (l'élément **LOCALNAMESPACEPATH**, ligne 2), et la liste des paramètres de l'appel (l'élément **IPARAMVALUE**, ligne 2).

L'élément **METHDOCALL** (lignes 7 à 12) permet d'appeler une méthode extrinsèque. Il permet de donner, de manière analogue à l'élément **IMETHODCALL**, le nom de la méthode, une référence vers l'objet cible de

l'opération (l'élément **LOCALINSTANCEPATH** pour les instances, et l'élément **LOCALCLASSPATH** pour les classes[23].), et la liste des paramètres de la méthode (l'élément **PARAMVALUE**).

7.6.3 Un exemple de représentation d'une requête

Dans cette section, nous donnons un exemple de représentation de l'opération CIM **GetClass**. L'exemple est extrait de la liste d'exemples donnés en annexes dans la spécification des opérations CIM [9].

```
1  < MESSAGE ID="36000" PROTOCOLVERSION="1.0" >
2    < SIMPLEREQ
3      < IMETHODCALL NAME="GetClass" >
4        < LOCALNAMESPACEPATH >
5          < NAMESPACE NAME="root"  />
6          < NAMESPACE NAME="cimv2"  />
7        < /LOCALNAMESPACEPATH >
8        < IPARAMVALUE NAME="ClassName" >
9          < CLASSNAME NAME="CIM_ComputerSystem"  />
10       < /IPARAMVALUE >
11     < /IMETHODCALL >
12   < /SIMPLEREQ >
13 < /MESSAGE>
```

FIG. 63 – *Exemple de représentation d'une opération CIM*

Dans cet exemple, l'élément racine **MESSAGE** donne l'identificateur du message (ligne 1). L'élément **IMETHODCALL** (ligne 3) permet de donner le nom de la méthode intrinsèque invoquée (ici `GetClass`). Dans l'arborescence de la représentation nous retrouvons une référence vers l'espace de nommage cible de la requête (élément **LOCALNAMESPACEPATH**, lignes 4 à 7), suivie de la liste des paramètres de la méthode (ici seul le nom de la classe requise est donné, lignes 8 à 10).

7.6.4 Représentation d'une réponse

D'une manière analogue à la représentation des appels de méthodes, le DMTF a défini 2 éléments pou la représentation des réponses : **IMETHODRESPONSE** et **METHODRESPONSE** correspondant respectivement à une réponse à un appel de méthode intrinsèque et à une réponse à un appel de méthode extrinsèque.

Pour le cas des réponses intrinsèques, l'élement **IMETHODRESPONSE** contient soit l'élément de réponse à la requête **IRETURNVALUE**, soit un élément d'erreur **ERROR** permettant de retourner, en cas d'échec de l'appel, des informations sur les causes de l'échec tel que le type d'erreur et une description textuelle de l'erreur. L'élément **METHODRESPONSE** utilisé pour les réponses à des méthodes extrinsèques est spécifié de manière analogue tel que l'illustre la figure 64.

7.6.5 Un exemple de représentation d'une réponse

Dans cet exemple, nous donnons la représentation d'une réponse à la requête donnée précédemment (figure 62).

Les éléments imbriqués permettent de spécifier les paramètres de la réponse : l'identificateur de message (ligne 1), le type de message (réponse simple, ligne 2), le nom de la méthode (ligne 3) et le paramètre de retour(lignes 4 à 8).

7.7 L'encapsulation HTTP

Afin de réaliser le transport de l'information CIM entre deux entités WBEM, le DMTF a opté pour l'utilisation du protocole HTTP [15] étendu [16].

Le standard proposé [9] consiste à encapsuler la description XML des opérations CIM (opérations de requête ou de réponse, voir section 7.6) dans le corps d'une unité de données HTTP. Cette encapsulation des opérations CIM est schématisée dans la figure 66.

23. Ceci est dû à la possibilité dans CIM de définir des méthodes de classe via le qualifieur standard `Static`

Reponse à une méthode instrinsèque

```
<!ELEMENT IMETHODRESPONSE
    ( ERROR | IRETURNVALUE )
>
<!ATTLIST IMETHODRESPONSE
    NAME    CDATA    #REQUIRED
>
```

Réponse à une méthode extrinsèque

```
<!ELEMENT METHODRESPONSE
    ( ERROR | ( RETURNVALUE ? , PARAMVALUE * ) )
>
<!ATTLIST METHODRESPONSE
   NAME    CDATA    #REQUIRED
>

<!ELEMENT ERROR EMPTY >
<!ATTLIST ERROR
    CODE        CDATA    #REQUIRED
    DESCRIPTION CDATA    #IMPLIED
>
```

FIG. 64 – *Représentation d'une réponse à une méthode*

```
< MESSAGE ID="36000" PROTOCOLVERSION="1.0" >
  < SIMPLERSP >
    < IMETHODRESPONSE NAME="GetClass" >
      < IRETURNVALUE >
        < CLASS NAME="CIM_ComputerSystem" SUPERCLASS="CIM_System" > ...
        < /CLASS >
      < /IRETURNVALUE >
    < /IMETHODRESPONSE >
  < /SIMPLERSP >
< /MESSAGE >
```

FIG. 65 – *Exemple de représentation d'une réponse*

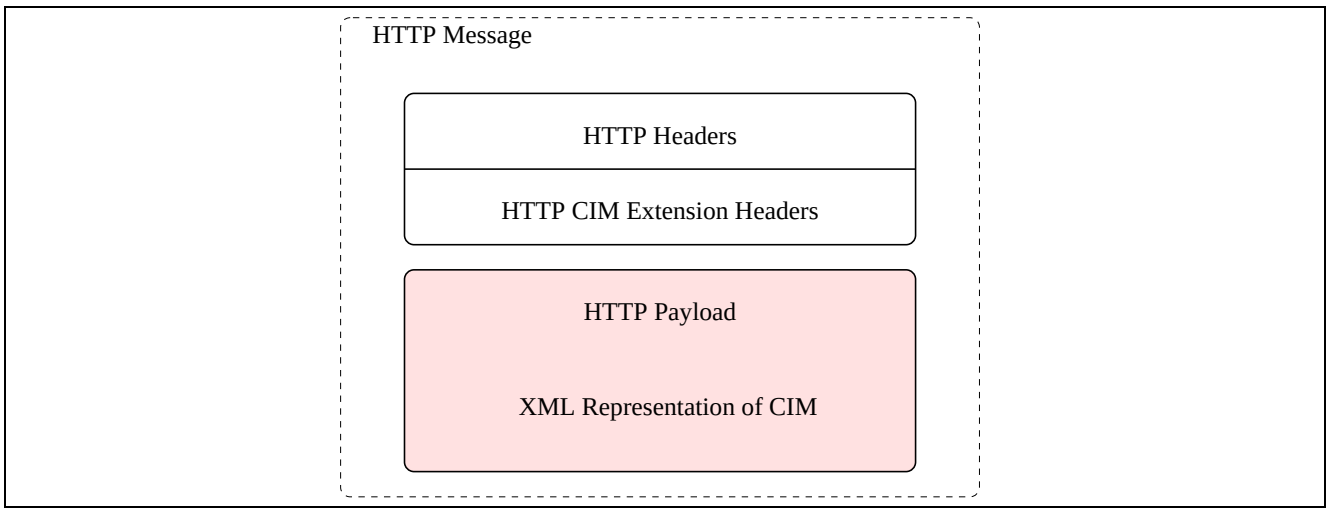


FIG. 66 – *Format des messages HTTP échangés dans WBEM*

Les messages HTTP échangés sont soit du type **M-POST** si le serveur HTTP supporte le mécanisme d'extension [16], soit du type **POST** simple le cas échéant.

L'entête du message HTTP comporte deux parties : les entêtes d'un message `POST` définies par le protocole HTTP, et les entêtes CIM définies dans le cadre de l'extension de HTTP pour le transport de l'information CIM. Les entêtes CIM sont au nombre de 6 :

1. **CIMOperation** permet de donner le type de l'opération CIM (invocation de méthode ou réponse) ;
2. **CIMProtocolVersion** permet de donner la version de l'encapsulation de CIM dans HTTP ;
3. **CIMMethod** n'est utilisé que pour les opérations d'invocation simples (`SIMPLEREQ`), et donne dans ce cas le nom de la méthode invoquée ;
4. **CIMObject** n'est utilisé que pour les opérations d'invocation simples et il donne dans ce cas le nom de l'objet cible de la méthode invoquée ;
5. **CIMBatch** permet d'indiquer que le paquet contient une requête multiple ;
6. **CIMError** utilisé pour les réponses comportant une erreur, il permet de donner le type de l'erreur ;

7.8 Résumé

Dans cette section nous avons présenté les représentations des informations CIM (objets et requêtes) dans le langage XML, tels qu'ils ont été spécifiés par le DMTF dans la version 2.0.

Concernant la représentation de données CIM, la spécification du DMTF repose sur une représentation du méta-modèle CIM par des éléments XML. Ceci permet de générer une représentation uniforme des modélisations CIM aussi bien pour des classes que pour des objets gérés (instances).

Une stratégie équivalente de traduction a été utilisée pour la représentation des messages échangés entre deux entités WBEM, ce qui présente l'avantage d'être extensible pour de nouvelles opérations : les éléments utilisés sont génériques à tout type d'appel de méthode.

Cette représentation est complète, dans le sens ou elle couvre toute l'information CIM, mais malheureusement "efface" toute la sémantique du modèle : la validation sémantique des représentations des objets gérés vis-à-vis des définitions de classes, nécessite une application spécifique supplémentaire en plus des outils de validation XML existants.

8 Intégration d'autres approches de gestion

L'approche WBEM plus que toute autre approche de gestion prend en compte l'existence et le déploiement d'autres approches de gestion. La prise en compte signifie que les objets gérés issus d'approches différentes que WBEM soient vus comme des objets "normaux", c'est-à-dire specifiés dans le formalisme MOF, présents dans une base d'informations de gestion d'un serveur CIMOM, et accessibles via l'interface CIM.

Pour mettre cela en place, il faut pouvoir exprimer dans le modèle CIM, les spécifications des objets gérés issus d'autres approches de gestion (DMI, SNMP, OSI, ...). On doit donc dans un premier temps, réaliser l'intégration des modèles de l'information.

Plusieurs solutions pour l'intégration d'autres modèles de l'information sont proposées. Ces solutions sont détaillées dans les sections suivantes.

8.1 Le mapping de technique

Le mapping de technique consiste à décrire le méta-modèle de l'approche que l'on désire intégrer à l'aide des composants du méta-modèle CIM.

L'activité principale pour ce type de mapping consiste à :

- identifier les composants du méta-modèle source ;
- décrire ces composants sources à l'aide des éléments du méta-modèle CIM ;
- reformuler les spécifications du modèle source en utilisant les descriptions CIM du méta-modèle source.

Si, par exemple, on voulait appliquer cette stratégie de traduction pour l'intégration du modèle OSI dans le modèle CIM, on pourrait définir l'approche suivante :

- spécifier une classe CIM pariculière (`OSI_ManagedObjectClass`) capable de décrire une `classe d'objet géré` issue du méta-modèle OSI (`ManagedObjectClass` dans GDMO) ;
- exprimer toute définition de classe GDMO en utilisant la classe OSI_ManagedObjectClass.

Cette stratégie de traduction est trop générique car basée sur de simples descriptions dans le modèle destination. Elle necessite un effort supplémentaire pour une validation sémantique et par conséquent reste peu utilisée.

8.2 Le mapping de recast

Le mapping de recast est légèrement différent du mapping de technique. Dans cette approche, on définit pour chaque élément du méta-modèle source, une correspondance avec un (ou plusieurs) composant(s) du méta-modèle destination.

Pour reprendre l'exemple de l'intégration du modèle OSI dans le modèle CIM, la stratégie de recast consisterait à définir des règles de correspondance telles que la correspondance entre une définition de classe OSI et une définition de classe CIM, une définition d'attribut OSI et une définition de propriété CIM, et ainsi de suite pour tous les éléments du méta-modèle OSI, comme l'illustre la figure 67.

Classe OSI

```
network  MANAGED OBJECT CLASS
   DERIVED FROM top
   CHARCTERIZED BY
    networkPackage  PACKAGE
       ATTRIBUTES networkId GET;;;
 REGISTERED AS {x.y.z}
```

Classe CIM

```
 class OSI_network  : OSI_top
  {
      [Read(true)]
    string networkId;
} ;
```

FIG. 67 – *Un exemple de traduction de recast*

Ce type de mapping, bien que parfois difficile à mettre en œuvre (la difficulté consiste à trouver des correspondances entre les éléments des deux méta-modèles), est assez intuitif et reste, de ce fait le plus souvent utilisé. Il est notamment utilisé pour l'intégration de l'approche SNMP dans l'OSI (IIMC [12]) ainsi que pour l'intégration SNMP/OSI dans une architecture CORBA (JIDM [25]). Dans WBEM, ce type d'intégration est utilisé pour la génération de proxys DMI ainsi que pour l'intégration SNMP.

8.3 Le mapping de domaine

Le mapping de domaine ne se base plus sur les méta-modèles. En effet, il permet pour une spécification d'un modèle de l'information de construire au coup par coup, les objets correspondants dans les schémas du CIM. Ces schémas peuvent être le schéma de base, le schéma commun ou un schéma spécifique.

Très recherchée par les spécialistes du domaine de la gestion de réseaux, cette technique de traduction reste cependant très peu utilisée parce qu'elle est fastidieuse à élaborer à cause d'un grand nombre d'objets à traduire sans aucune automatisation possible. Cependant, elle est utilisée dans l'approche DEN [22] (*Dircetory-Enabled Networks*) ou des mappings de domaine sont mis en oeuvre pour des objets du modèle réseau et du modèle de politiques vers des objets LDAP [27] (*Lightweight Directory Access Protocol*).

8.4 Les mappings existants

Il existe relativement peu de travaux aboutis sur l'intégration des approches existantes dans WBEM. Il n'existe à ce jour qu'une intégration du formalisme MIF et des spécifications DMI dans WBEM ainsi qu'une intégration SNMP. Ces deux intégrations basées sur un mapping de recast sont présentées dans cette section.

8.4.1 Intégration DMI

DMI [5] (*Desktop Management Interface*) est le modèle de gestion proposé par le DMTF depuis 1994 pour la gestion des postes de travail (gestion des composants matériels et logiciels). Depuis 1998, date de l'adoption

de l'initiative WBEM et de la normalisation de CIM, le DMTF a tout naturellement proposé un mapping CIM/DMI de type `recast` pour permettre aux applicationx WBEM d'accèder aux agents DMI (*DMI Service Provider*). Les règles de production génériques définies dans ce mapping sont les suivantes :

- toute définition de composant DMI (entité de base pour la modélisation des éléments logiques ou physiques d'un système) est traduite en une définition de classe CIM ;
- toute définition de groupe DMI (entité organisationnelle permettant de regrouper un ensemble d'attributs d'un composant donné sous un même nom) est traduite en une définition de classe CIM ;
- toute définition d'attribut DMI est traduite en une définition de propriété CIM ;
- toute définition d'attribut clef DMI est traduite en une définition de propriété clef CIM.

Les spécifications données ci-dessous illustrent l'application de ces règles de traduction pour générer des objets CIM à partir de spécifications MIF (*Management Information Format*: langage de spécification des éléments gérés dans l'approche DMI). Ces spécifications sont tirées d'un exemple donné dans le document de noramlisation de CIM [8].

La spécification MIF

```
Start Group
    Name = "ComponentID"
    Class = "DMTF|ComponentID|001"
    ID = 1
    Description = "This group defines the attributes common to all components"
    Start Attribute
        Name = "Manufacturer"
        ID = 1
        Access = Read-Only
        Type = DisplayString(64)
        Value = ""
    End Attribute
End Attribute
```

La spécification CIM

```
    [ Name("ComponentID"), ID(1), Description(
     "This group defines the attributes common to all components")]
class DMTF|ComponentID|001 {
        [ ID(1), Read, MaxLen(64)]
    string Manufacturer;
};
```

Le mapping est intéressant en tant que tel, mais le point essentiel illustré ici est, à notre avis, l'apport des qualifieurs dans les traductions des modèles : ils offrent à CIM la souplesse nécessaire pour prendre en compte les spécificités des autres modèles de gestion. Ici, le DMTF a défini de nouveaux types de qualifieurs pour représenter certains éléments de MIF (par exemple, le qualifieur `ID` est utilisé pour représenté le méta-attribut ID du modèle MIF).

8.4.2 Intégration SNMP

L'intégration SNMP doit permettre à tout superviseur WBEM d'accéder à des objets SNMP via un agent CIMOM et suivant le modèle CIM. Il n'existe aujourd'hui officiellement aucune norme ni standard définissant l'intégration de SNMP dans CIM. Cepedant, l'implantation du CIMOM de Microsoft distribuée sur Internet en 1998 intègrait une telle fonction et ces algorithmes et représentations fournissent une excellente illustration des mécanismes de CIM utiles à l'intégration et pourraient servir de base à une future standardisation au DMTF. C'est pourquoi nous les présentons dans cette section.

Le type de mapping retenu est un mapping de recast. L'objectif pour les modèles de l'information est donc pour chaque composant du méta-modèle SNMP (SMI), d'identifier un composant du méta-modèle CIM capable de le représenter. Les choix qui ont été réalisés sont très proches de ceux définis dans le mapping SNMP/OSI et sont les suivants:

- tout groupe SNMP est traduit en une classe CIM;

- tout objet SNMP est traduit en un attribut CIM;
- tout tableau SNMP est traduit en une classe CIM;
- toute caractéristique d'un composant SNMP (type,statut, convientionn textuelle,...) est représenté par un qualifieur.

Afin de permettre la mise en correspondance de types et de caratéristiques, l'approche d'intégration définit pour les attributs CIM représentant des objets SNMP un certain nombre de qualifieurs. Ceux-ci sont listés dans le tableau 68

Nom	Type	Sémantique
description	string	le pendant de la clause description en SNMP.
name	string	le nom de l'objet snmp
status	string	la valeur du champ status dans SNMP
object_identifier	string	l'identificateur d'enregistrement OSI de l'objet
object_syntax	string	correspondant de la clause SYNTAX de SNMP
cim_type	string	représentation textuelle du type CIM utilisé
textual_convention	string	convention textuelle définie pour ce type dans SNMP
encoding	string	type SNMP utilisé pour l'encodage de la valeur
fixed_length	uint32	taille de la propriété
variable_length	string	taille variable (min..max) de la propriété
variable_value	string	intervalle des valeurs possibles
enumeration	string	liste séparée de virgules des valeurs possibles
bits	string	définition des valeurs énumérées
display_hint	string	pendant de la clause DISPLAY_HINT des conventions textuelles de SNMP
key	bool	vrai si l'attribut est une valeur clef d'un tableau
key_order	uint32	position de la clef dans la liste des clefs
virtual_key	bool	vrai si la clef doit être calculée depuis d'autres valeurs dans l'agent SMNP
read	bool	l'attribut est accessible par un GET
write	bool	l'attribut est accessible pour une affectation (SET)
defval	string	valeur par defaut dans l'agent SNMP
référence	string	l'attribut contient une réf'erence vers un autre objet
units	string	pendant de la clause UNITS de SNMP. Spécifie l'unité dans laquelle est exprimée la valeur de l'attribut

FIG. 68 – *Les qualifeurs de mapping SNMP/CIM*

Concrètement, l'on retrouve sous forme de qualifieur d'attribut, tous les composants d'une spécification de type d'objet dans SMI SNMP.

Pour les classe CIM générées à partir de groupes SNMP, les qualifieurs prédéfinis dans le mapping sont listés dans le tableau 69.

Ces qualifieurs sont associés à chaque classe CIM générée depuis une spécification SNMP.

Afin d'illustrer la traduction SNMP dans des modèles CIM, nous présentons deux exemples. L'un est une traduction d'un groupe standard de la MIB-II: le groupe system; le second est un tableau également issu de la MIB-II: la table des statistiques pour UDP.

La figure 70 comprend un sous-ensemble de la spécification du groupe system de la MIB-II.

On retrouve dans cette spécification, 3 objets SNMP qui sont le descripteur du système, le compteur indiquant la durée écoulée depuis le dernier redémarrage ainsi que la personne à contacter pour ce système.

La figure 71 comporte la spécification CIM issue de la traduction de ces objets SNMP.

Le groupe system est traduit en une classe CIM (classe `SNMP_RFC1213_MIB_system`). Le nom de celle-ci est le résultat de la concaténation de l'identificateur de RFC, le mot clef MIB et le nom du groupe. La classe possède les qualifieurs énumérés dans la table 69. Parmi ces qualifieurs on trouve le nom du module dans lequel la classe est définie (`module_name("RFC1213-MIB")`) ainsi que l'identificateur du groupe dans le module (`group_objectid("1.3.6.1.2.1.1")`). Le qualifieur singleton indique qu'une seule instance de cette classe ne

Nom	Type	Sémantique
description	string	la description de la classe. En général celle-ci est vide pour un groupe et comprend la concaténation des descriptions du tableau et de son élément SNMP pour les classes issues d'un mapping de tableau.
module_name	string	le nom informel du groupe
group_objectid	string	l'OID définissant le préfixe du groupe
module_imports	string	la liste des modules importés dans ce module
singleton	bool	précise que la classe CIM ne peut être instanciée qu'une seule fois. Ceci est vrai pour tous les groupes pas pour les tableaux car il y aura autant d'instance d'une classe tableau que d'éléments dans ce tableau.

FIG. 69 – *Les qualifeurs de classes pour l'intégration SNMP/CIM*

```
sysDescr OBJECT-TYPE
        SYNTAX DisplayString (SIZE (0.255))
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "A textual description of the entity..."
:= {system 1}
...
sysUpTime OBJECT-TYPE
        SYNTAX TimeTicks
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The time (in hundreds of a second) since the ...."
:= {system 3}

sysContact OBJECT-TYPE
        SYNTAX DisplayString (SIZE (0..255))
        ACCESS read-write
        STATUS mandatory
        DESCRIPTION "The textual identification of the contact person ..."
:= {system 4}
```

FIG. 70 – *Un sous-ensemble du groupe system de la MIB-II*

```
[
description(""),module_name("RFC1213-MIB"),singleton,
group_objectid("1.3.6.1.2.1.1"), dynamic, provider("MS_SNMP_INSTANCE_PROVIDER")
]
class SNMP_RFC1213_MIB_system: SnmpObjectType

 [
  textual_convention("DisplayString"),encoding("OCTETSTRING"),object_syntax("DisplayString"),
  variable_length("0..255"), object_identifier("1.3.6.1.2.1.1.1"), read,
description("A textual description of the entity..."), status("mandatory")
 ] string sysDescr;
 [
  textual_convention("TimeTicks"),encoding("TimeTicks"),object_syntax("TimeTicks"),
  object_identifier("1.3.6.1.2.1.1.3"), read,
description("The time (in hundreds of a second) since the ...."), status("mandatory")
 ] string sysUpTime=0;

 [
  textual_convention("DisplayString"),encoding("OCTETSTRING"),object_syntax("DisplayString"),
  variable_length("0..255"), object_identifier("1.3.6.1.2.1.1.4"), read, write,
description("The textual identification of the contact person ..."), status("mandatory")
 ] string sysContact;
```

FIG. 71 – *La traduction CIM du sous-ensemble du groupe system de la MIB-II*

peut exister dans cet espace de nommage. Le qualifieur dynamic indique que les valeurs sont obtenues par accès direct à l'agent SNMP. L'identificateur du provider assurant cette liaison dynamique est fournie dans le qualifieur provider.

Tout objet SNMP du groupe est traduit en un attribut. La table de correspondance de types est donnée dans la table 72.

Type SNMP	**type CIM**
INTEGER	sint32
OCTET STRING	string
OBJECT	string
NULL	-
IpAddress	string
Counter	uint
Gauge	string
TimeTicks	uint
Opaque	string
NetworkAddress	string
DateAndTime	string
DisplayString	string
MAcAddress	string
PhysAddress	string
SnmpUDPAddress	string
SnmpOSIAddress	string
SnmpIpxAddress	string

FIG. 72 – *Les correspondances de types SNMP/CIM*

À chaque mapping de type s'ajoute l'insertion de trois qualifieurs d'attribut comme défini dans la table 68. Ces qualifieurs sont celui définissant les conventions textuelles (`textual_convention`), celui spécifiant l'encodage ASN.1 utilisé entre le provider et l'agent SNMP pour cet attribut (`encoding`) et finalement le qualifieur définissant la syntaxe ASN.1 de l'attribut (`object_syntax`).

On retrouve également pour chaque attribut les qualifieurs d'accès (read,write), les spécialisation de types (restriction de longueur ou d'espace de valeurs comme par exemple le qualifieur `variable_length("0..255")` pour l'attribut syscontact), les qualifieurs de description, d'identificateur d'objet ainsi que le statut.

Le second exemple SNMP est donné dans la figure 73 et représente la table UDP de la MIB-II.

```
udpTable OBJECT-TYPE
        SYNTAX SEQUENCE OF UdpEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "A table containing UDP listener information."
::= udp 5

udpEntry OBJECT-TYPE
        SYNTAX UdpEntry
        ACCESS not-accessible
        STATUS mandatory
        DESCRIPTION "Information about a particular UDP listener"
        INDEX udpLocalAddress, udpLocalPort
::= udpTable 1

UdpEntry ::= SEQUENCE
        {
          udpLocalAddress  IpAddress,
          udpLocalPort INTEGER (0..65535)
        }

udpLocalAddress OBJECT-TYPE
        SYNTAX IpAddress
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The local IP address ..."
::= udpEntry 1

udpLocalPort OBJECT-TYPE
        SYNTAX INTEGER (0..65535)
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION "The local port number for this UDP listener."
::= udpEntry 2
```

FIG. 73 – *Un exemple de tableau en SNMP*

La spécification CIM associée est donnée dans la figure 74. Les mécanismes de traduction sont proches de ceux des groupes. Les principales différences résident dans l'absence du qualifieur `singleton` dans la spécification de classe (plusieurs instances de cette classes peuvent exister dans l'espace de nommage) et dans la prise en compte du qualifieur `key` pour identifier les attributs servant d'index dans le tableau.

Dans l'ensemble des exemples ci-dessus, on retrouve l'intérêt des qualifieurs extensibles de l'approche CIM. En effet, la définition de qualifieurs par domaine d'intégration permet facilement d'intégrer dans des spécifications CIM des informations relatives au domaine intégré, informations qui facilitent la réalisation des agents d'intégration à proprement parler.

Dans l'implantation testée, un compilateur SNMP vers CIM était fourni. Une fois les spécifications CIM générées, l'on pouvait créer un provider capable de réaliser la passerelle. Celui-ci était réalisé en instanciant un

```
  [ description("A table containing UDP listener information."
    "Information about a particular UDP listener"), module_name("RFC1213-MIB"),
    group_objectid("1.3.6.1.2.1.7"), dynamic, provider("MS_SNMP_INSTANCE_PROVIDER") ]
class SNMP_RFC1213_MIB_udpTable: SnmpObjectType
{
     [ textual_convention("IpAddress"), encoding("IpAddress"), object_syntax("IpAddress"),
       object_identifier("1.3.6.1.2.1.7.5.1.1"), read,
       description("The local IP address ..."), status("mandatory"), key, key_order(1) ]
   string udpLocalAddress="0.0.0.0";
     [ textual_convention("INTEGER"), encoding("INTEGER"), object_syntax("INTEGER"),
       variable_value("0..65535"), object_identifier("1.3.6.1.2.1.7.5.1.2"), read,
       description("The local port number for this UDP listener."), status("mandatory") ]
   sint32 udpLocalPort=0;
}
```

FIG. 74 – *La traduction CIM du tableau SNMP*

espace de nommage spécifique par agent SNMP. Cet espace de nommage possède les qualificateurs spécifiques suivants:

- AgentAddress : adresse IP de l'agent SNMP correspondant;
- AgentTransport : le protocole utilisé pour communiquer avec l'agent;
- Agent(Read/Write)CommunityName : les noms de communauté pour les opérations de lecture/affectation;
- AgentRetry(Count/Timeout) : nombre d'essai maxmial pour une requête, temps maximal d'attente avant réémission;
- AgentVarBindsPerPDU : nombre maximal de variables par paquet SNMP
- AgentFlowControlWindowSize : taille de la fenêtre d'anticipation de requêtes SNMP (en nombre de requêtes);
- AgentSnmpVersion : version du protocole à utiliser pour accéder à l'agent.

Au sein de cet espace de nommage qui modélise l'agent, on retrouvera les spécifications des mappings ainsi que les instances (de manière dynamique). Cet espace de nommage est rempli par deux providers: l'un pour les spécifications, l'autre pour les instances. Ce dernier est en fait le véritable proxy-SNMP.

8.5 Intégration inverse

Si WBEM se veut une approche d'intégration, il est clair qu'un mapping inverse vers des approches telles que l'OSI est également nécessaire. Les seuls résultats disponibles dans ce sens sont ceux menés au sein du projet RESEDAS de l'INRIA Lorraine. Les résultats aujourd'hui disponibles sont une intégration de recast de modèles CIM dans une approche OSI permettant la géneration automatique de proxys OSI/WBEM. En plus des algorithmes de mappings, des outils d'intégration ainsi qu'un agent d'adaptation générique sont disponibles [13, 14].

8.6 Résumé

Présentée comme une approche d'intégration, WBEM se doit de proposer des mécanismes pour le support d'approches telles que DMI et SNMP au minimum; OSI si l'on désire être exhaustif. Actuellement seule une intégration DMI est proposée dans le standard CIM et encore, celle-ci n'est présentée que par un exemple (voir intégration DMI dans ce chapitre). Pour l'intégration SNMP, seule la version beta de Microsoft distribuée en 1998 sur Internet proposait un toolkit d'intégration (voir section intégration SNMP) dans ce chapitre. Celui-ci est intégré dans la distribution actuelle de WMI (voir sec. 9.1). Au niveau du DMTF, il n'existe à ce jour aucun standard décrivant celle-ci.

Les deux approches de traduction présentées dans cette section montrent bien la capacité de CIM à intégrer d'autres modèles de l'information, et ce notamment, grâce à la souplesse apportée par la notion de qualifieur. En effet, l'extension du modèle CIM par la définition de nouveaux qualifieurs offre un moyen simple et efficace pour

l'intégration. Ceci est très intéressant, mais doit tout de même être utilisé avec beaucoup de précautions, dans la mesure où l'ajoût massif de qualifieurs propriétaires ne peut que nuire à l'interopérabilité entre les applications WBEM.

Il est regretable que, malgré les potentialités de CIM, le DMTF ne standardise pas à ce jour d'algorithmes d'intégration, démontrant ainsi la capacité d'intégration de l'approche. Cependant, les différents mappings proposés par certains industriels sont intéressants mais restent dans une approche de mapping de recast. Or, Il apparaît au travers de la définition de CIM que cette approche pourrait être extrêmement utile pour du mapping de domaine qui, malgré le fait que celui-ci n'est pas entièrement automatisable, est très prisé, car il permet une intégration beaucoup plus forte du point de vue sémantique des objets gérés. C'est dans ce sens que le projet RESEDAS développe aujourd'hui une activité de recherche.

9 Les implémentations de WBEM

Plusieurs produits CIM/WBEM existent à ce jour :

- **Windows Management Instrumentation (WMI)** implémentation complète de WBEM développée par Microsoft. Une présentation détaillée du produit est donnée dans la section 9.1 ;
- **Solaris WBEM Services** implémentation complète développée developpé par SUN Microsystems. Une présentation détaillée du produit est donnée dans la section 9.2 ;
- **SNIA CIMOM** implémentation OpenSource d'un CIMOM réalisé au sein du SNIA (`Storage Networking Industry Association`). Pour plus de détails sur le produit, nous renvoyons le lecteur au site Web SNIA (http://www.snia.org) ;
- **JMX WBEM** interface de programmation écrite en Java implantant les services de communication WBEM et développée dans le cadre du consortium JMX (`Java Management eXtentions`). La spécification de cette interface est donnée dans [17].

9.1 Microsoft Windows Management Instrumentation

Initialement appelée WBEM SDK, l'implémentation WBEM de Microsoft Corp. a été modifiée depuis la disparition du protocole HMMP. Actuellement Microsoft propose une implémentation WBEM comportant un gestionnaire d'objets CIM (CIMOM) avec une interface d'accès via COM/DCOM et un certain nombre de providers.

Dans cette section, nous présentons chacun de ces éléments. Dans un premier temps, nous donnons une vue globale de l'architecture de l'implémentation proposée par Microsoft, puis nous présentons brièvement le schéma CIM spécifié pour la gestion du monde Windows, l'interface d'accès au serveur CIMOM ainsi que les différents services proposés par WMI.

9.1.1 Architecture

L'implémentation de WBEM de Microsoft (WMI : Windows Management Instrumentation [23, 24, 1, 2, 3]) [24], permet le développement d'applications de gestion suivant une architecture WBEM. Dans sa distribution actuelle, WMI présente les composants schématisés dans la figure 75 :

- le gestionnaire d'objets CIM : composant de base de l'implémentation Microsoft, il maintient une base de l'information de gestion (`CIM Object Repository`), et offre une interface d'accès COM/DCOM à cette base ;
- les providers : Microsoft fournit dans son implémentation WBEM, quelques providers permettant l'acquisition de l'information de gestion relative aux différents éléments du domaine géré. Actuellement ces providers, que nous ne détaillerons pas dans ce rapport sont :

  `Win32 Provider` : gère les informations qui concernent le système d'exploitation, l'ordinateur et ses périphériques ;

  `WDM Provider` : gére des informations bas niveau du WDM (Windows Driver Model) ;

  `Event Log Provider` : permet d'accèder aux journaux générées par le système ;

  `Performance Counter Provider` : uniquement supporté par Windows 2000, il permet d'accèder aux valeurs des compteurs de performances ;

24. `http://www.microsoft.com/hwdev/WMI/`

`Active Directory Provider` : fonctionne comme une passerelle à toute l'information maintenue dans le service d'annuaire `Microsoft Active Directory Services` ;

`Windows Installer Provider` : permet d'accèder à toutes les informations sur les installations de logiciels dans le système ;

`SNMP Provider` : fonctionne comme une passerelle au monde SNMP. Il permet d'accèder à l'information maintenue par un agent SNMP et de la présenter aux utilisateurs de WMI dans le modèle CIM ;

- un service de sécurité : très limité dans la version actuelle de WMI, il ne comporte qu'un mécanisme d'authentification des clients du CIMOM. Une fois l'accès au CIMOM authentifié, le client n'a aucune restriction sur les objets maintenus dans la base de gestion CIM ;
- un service de notifications : encore inexistantes dans l'approche WBEM, les notifications d'évènements peuvent être manipulées à travers l'implémentation de Microsoft. En effet, WMI offre un service permettant de spécifier des évènements particuliers tels que des erreurs logicielles ou des défaillances matérielles, et de générer par la suite les notifications correspondantes à ces évènements. Ce mécanisme est présenté dans la section 9.1.2 ;
- un service de requêtes : WMI implante un service de requêtes basé sur un sous ensemble du langage SQL, baptisé `WMI Query Langage (WQL)` [25] ;

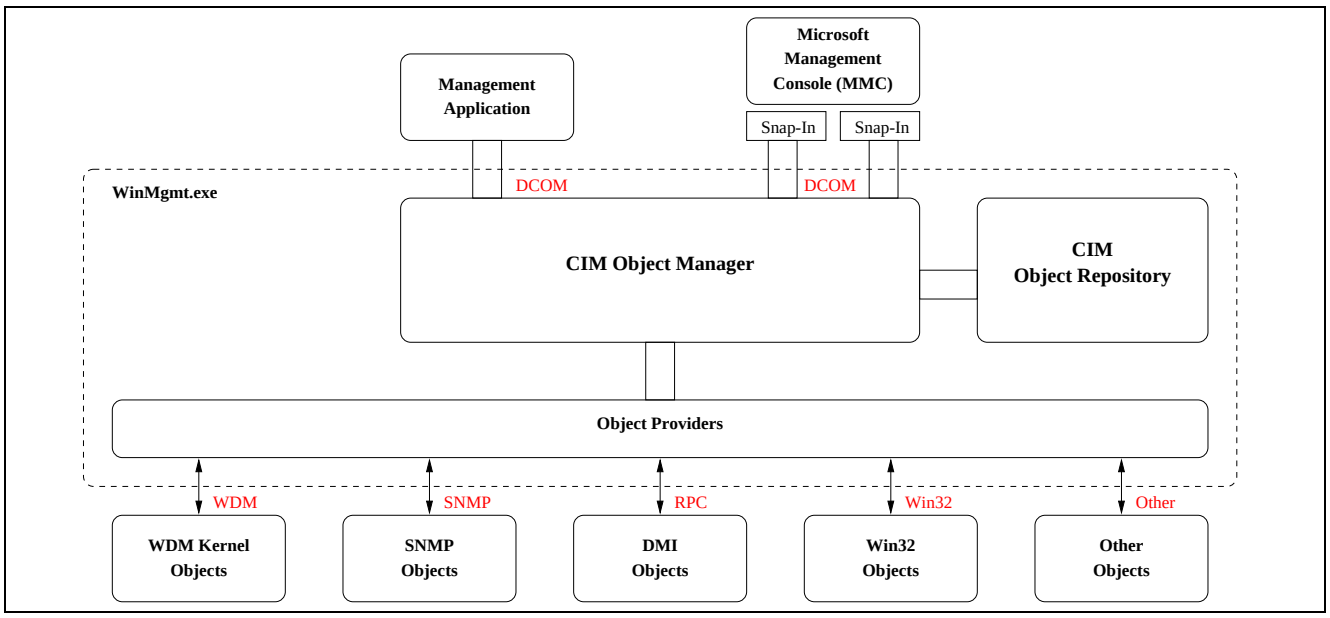


FIG. 75 – *Architecture de l'implémentation WMI*

9.1.2 La gestion des évènements

La distribution actuelle de WMI offre la possibité de souscrire et de recevoir des notifications d'évènements : Chaque application accédant au CIMOM, peut définir dans une première phase des conditions particulières pour l'émission de notifications. Ces conditions peuvent être de deux type : des évènements de niveau bas (défaillances matérielles par exemple) ou des évènement au niveau de la base de gestion du CIMOM (changement d'une valeur de propriété par exemple). Lors de l'occurence de l'évènement, le provider d'évènement inclu dans la distribution de WMI, notifie le CIMOM qui se charge de distribuer par la suite la notification à toutes les applications consommatrices de l'évènement. La souscription aux notifications et la définition des déclenchement de leurs émission sont spécifiées sous formes de filtres dans le langage de requête WQL (WMI Query Langage).

Ce modèle d'émission de notifications est pour le moment unique dans la communauté WBEM qui n'a toujours pas défini cette forme d'interaction malgré son importance dans le monde de la gestion.

25. Dernièrement proposé pour une standardisation au DMTF sous le nom de WBEM Query Langage

9.1.3 Le navigateur de MIB

Application particulière de WMI, le navigateur de MIB, appelé `CIM Studio`, permet à un utilisateur d'interroger la base de gestion du CIMOM et puis d'afficher les diffèrents objets existants dans les espaces de nommage maintenus par ce dernier. `CIM Studio` permet de visualiser aussi bien les définititons des classes que les instances de celles-ci contenues dans un espace de nommage du CIMOM. Le navigateur est implanté sous forme d'une application ASP (Active Server Pages) accessible depuis un navigateur Web (Microsoft Internet Explorer). La figure 76 donne un apperçu sur cette interface graphique.

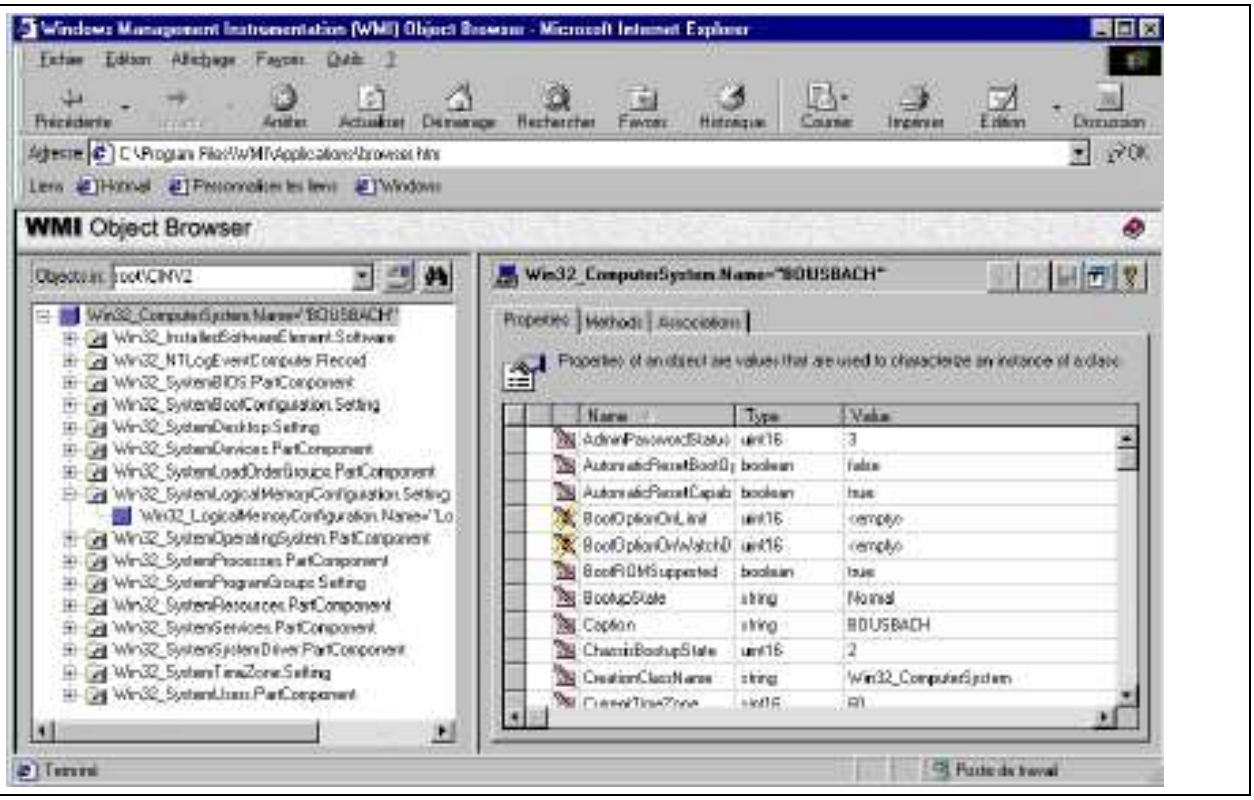


FIG. 76 – *L'interface graphique CIM Studio*

9.1.4 L'interface de programmation

L'interface de programmation du CIMOM définie par Microsoft est une interface C++ qui permet d'interroger le CIMOM à travers le protocole COM/DCOM. Les services de communication définis dans ce cadre reprennent les services déjà définis dans le cadre de la proposition initiale du protocole HMMP pour véhiculer l'information CIM.

Ces services d'accès aux objets maintenus dans la base de gestion du CIMOM sont définis comme des méthodes d'une interface COM **IWbemServices**. Dans la distribution actuelle de WMI, chaque service est implémenté par deux méthodes : une méthode pour une requête de type synchrone et une deuxième pour une requête de type asynchrone.

9.1.5 Le schéma Win32

Défini dans le fichier **"`cimwin32`"** inclu dans la distribution actuelle de WMI, le schéma Win32 spécifie une extension du schéma de base CIM du DMTF pour la gestion des mondes Windows 2000 et Windows NT.4. Dans cette section, nous présentons les principales caractéristiques du schéma.

1. Les extensions du schéma : afin de prendre en compte la gestion du monde Windows, WMI étend les schémas donnés par le DMTF par un ensemble de classes et associations dont les plus importantes sont :

`Win32_ComputerSystem` : elle étend la classe `CIM_UnitaryComputerSystem` du modèle système du DMTF pour modéliser la vision logique (voir distinction entre un élément logique et un élément physique dans le schéma de base du DMTF, sec. 5) d'un ordinateur sous un environnement Windows ;

`Win32_Service` : elle hérite de `Win32_BaseService`, qui elle même hérite de `CIM_Service` définie dans le modèle de base du DMTF. Elle permet de représenter un service implanté dans le système tel qu'un client DHCP (`Dynamic Host Configuration Protocol`) sous Windows NT ou Windows 2000 ;

`Win32_Process` : elle hérite de `CIM_Process` du modèle système du DMTF, et permet de modéliser un processus dans un environnement Windows ;

`Win32_OperatingSystem` : elle étend la classe `CIM_OperatingSystem` du modèle système du DMTF, pour modéliser les spécificités d'un système Windows ;

`Win32_NetworkProtocol` : elle hérite directement de `CIM_LogicalElement` du schéma de base du DMTF, et modélise un protocole réseau ;

`Win32_LogicalDisk` : elle étend la classe `CIM_LogicalDisk` du modèle de device du DMTF, et permet de représenter les espaces disques sous Windows ;

2. Les qualifieurs spécifiques : WMI définit, en plus des qualifieurs standards du DMTF, un ensemble de qualifieurs spécifiques au schéma Win32 tels que :

   `CIM_key` : associé aux propriétes, il spécifie que celles-ci forment une clef d'identification pour des instances du schéma CIM ;

   `AMENDMENT` : associé aux définitions de classes, il spécifie que celles-ci peuvent être traduites dans d'autres langues (`Locales`) et maintenues dans un espace de nommage spécifique à la langue en question ;

   `Locale` : permet de spécifier la langue de la définition originelle de l'élément ;

   `UUID` : associé à une définition de classe, il permet de donner un identificateur universel unique à celle-ci (`Universally Unique IDentifier`) ;

   `Dynamic` : permet de spécifier les classes dont les instances sont créées de manière dynamique par un provider. La classe à laquelle est associée ce qualifieur, doit aussi spécifier le provider qui en gère les instances, et ce par instanciation du qualifieur `Provider` ;

   `Implemented` : associé à une méthode, il spécifie que celle-ci possède une implémentation dans un provider ;

   `Deprecated` : associé à une propriété, il spécifie que celle-ci est remplacée par une autre propriété dans la dernière version du modèle ;

   L'opération d'ajoût d'une définition de qualifieur est autorisée dans CIM, mais reste déconseillée dans la mesure où elle nuit à l'interopérabilité des applications WBEM. Ceci est d'autant plus vrai quand les nouvelles définitions ont déjà leurs équivalents dans les schémas standards du DMTF. Par exemple, le qualifieur `CIM_Key` existe déjà en CIM (qualifieur standard `Key`), ou encore le qualifieur `UUID` est redondant par rapport au schéma du nommage universel des classes CIM (concaténation du nom du schéma avec le nom de la classe).

3. Les directives de compilation supplémentaires : de même que pour les qualifieurs, Microsoft a rajouté dans le fichier `MOF` définissant son modèle quelques directives de compilations supplémentaires (`#pragma autorecover` et `#pragma classflags`). Ceci est autorisé par la norme mais fortement déconseillé, dans la mesure ou il s'oppose à l'interopérabilité pour les mécanismes *import* et *export* entre différentes plates-formes WBEM.

9.2 Solaris WBEM Services

`Solaris WBEM Services` matérialise le support de SUN Microsystems pour le modèle de gestion WBEM annoncé en 1999. Cette implémentation de WBEM, téléchargeable depuis le WEB [26], s'installe sur les systèmes `Solaris 2.6`, `Solaris 7` ainsi que `Solaris 8`. Dans cette section nous en présentons les principales caractéristiques. Pour plus de détails sur le produit, nous renvoyons le lecteur à [19] [18].

26. http://www.sun.com/software/solaris/wbem

9.2.1 Architecture

La figure 77 présente les différents composants de l'implémentation WBEM de SUN Microsystems. Entièrement réalisés en Java, ces composants sont :

- `SUN WBEM User Manager` : composant logiciel permettant de gérer les accès utilisateurs au CIMOM. L'application offre la possibilité de définir les droits d'accès des utilisateurs à l'information maintenue dans les espaces de nommage gérés par le CIMOM ;
- `WBEM Log Viewer` : composant logiciel permettant la visualisation des données de "`log`". Ce service s'appuye sur les classes `Solaris_LogRecord` et `Solaris_LogService` définies dans le schéma Solaris. Ce classes sont détaillées dans la section 9.2.3 ;
- `MOF Compiler` : permet de générer à partir d'un fichier de spécifications MOF, les classes Java correspondantes au modèle CIM et de les rajouter à la base de gestion du CIMOM ;
- L'interface de programmation client : écrite en Java, elle permet de développer des applications WBEM. Pour cela, elle implante deux protocoles de communication : les RMI Java, et le protocole HTTP/XML proposé comme standard de communication WBEM par le DMTF ;
- le gestionnaire d'objets CIM (`CIMOM`) ;
- le répertoire d'objets CIM (`CIM Repository`) ;
- l'interface provider : elle permet de développer des providers pour le CIMOM ;
- `Solaris Provider` : donne accès aux ressources gérées sur un système Solaris définies dans le schéma Solaris. Il communique avec les ressources via une inteface native (JNI : Java Native Inteface).

En plus de l'implémentation d'une architecture WBEM comprenant un CIMOM ainsi que sa base de gestion associée, la distribution actuelle de `Solaris Services` offre quatre services supplémentaires que nous présentons dans les sections qui suivent.

9.2.2 Le service de sécurité

L'implémentation de SUN Microsystems, comprend dans sa distribution actuelle, un support pour la gestion des accès au serveur CIMOM. Le service proposé comporte deux mécanismes de sécurisation des interactions client/serveur :

- l'authentification : c'est la procédure d'identification des clients avant d'établir les connexions avec le serveur CIMOM. Dans la distribution actuelle, cette authentification s'appuye sur le système d'authentification des utilisateurs du système Solaris ;
- l'autorisation : permet de donner les droits d'accès à l'information pour chaque utilisateur. L'autorisation implantée dans la version actuelle du produit est très limitée : elle ne comprend qu'une autorisation par espace de nommage, i.e. un droit d'accès à un espace de nommage offre le même droit d'accès à tous les objets maintenus dans cet espace de nommage. Le service d'autorisations s'appuye sur 3 classes définies dans le schéma `Solaris` et maintenues dans l'espace de nommage **/root/security** reservé aux données utilisées par le CIMOM pour la gestion de la sécurité :

  `Solaris_Acl` : classe de base pour la définition des droits d'accès. Elle comprend un unique attribut `capability` permettant de donner un droit d'accès. Les valeurs possibles pour cet attribut sont `r` pour un droit d'accès en lecture simple, `rw` pour un droit d'accès en lecture et en écriture, `w` pour un droit d'accès en écriture simple et `none` pour une interdiction complète à l'information.

  `Solaris_UserAcl` : elle permet de modéliser un droit d'accès d'un utilisateur sur un espace de nommage. Elle hérite de `Solaris_Acl` et définit en plus deux attributs clefs : `nspace` pour donner le nom de l'espace de nommage et `username` pour donner le nom de l'utilisateur. Chaque instance de `Solaris_UserAcl`, maintenue dans l'espace de nommage `/root/security`, donne alors le droit d'accès d'un utilisateur sur un espace de nommage. De plus, pour un utilisateur donné et un espace de nommage donné, il ne peut exister qu'une unique instance de `Solaris_UserAcl`.

  `Solaris_NamespaceAcl` : elle permet de modéliser les droits d'accès de tous les utilisateurs sur un espace de nommage. Elle hérite de `Solaris_Acl` et définit en plus un attribut clef : `nspace` donnant le nom de l'espace de nommage pour lequel est défini le droit d'accès.

  La politique de gestion des autorisations recommandée par SUN, consiste à fixer les droits d'accès sur un espace de nommage en premier lieu (un droit d'accès en lecture simple par exemple), et puis de rajouter des droits supplémentaires aux différents utilisateurs (un accès en lecture et en écriture pour le gestionnaire par exemple).

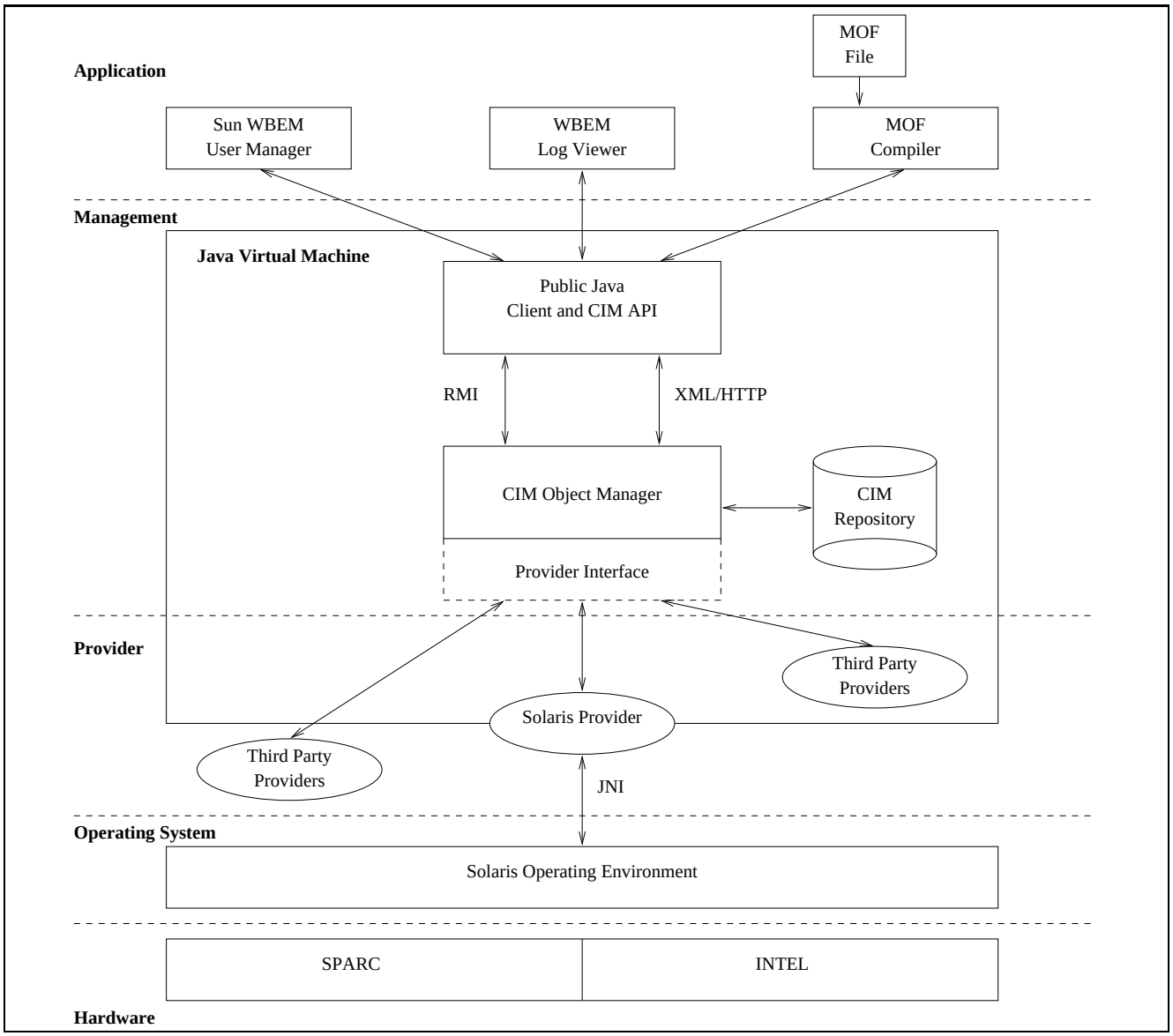


FIG. 77 – *Architecture de l'implémentation WBEM de SUN Microsystems*

9.2.3 Le service de log

Le service de log proposé dans l'implémentation WBEM de SUN Microsystems est présenté comme une application WBEM particulière. Il consiste d'une part à archiver les erreurs et autres évènements générés lors du déroulement d'une application, et d'autre part à offrir une interface d'accès à ce types de données dans le but de déterminer les causes des erreurs ou des évènements particuliers ayant eu lieu.

L'application de log est basée sur les classes définies dans le schéma `Solaris` :

- `Solaris_LogRecord` : elle est utilisée pour modéliser une donnée enregistrée dans un fichier de log. Lorsqu'un évènement particulier est détecté par une application ou un provider, une instance de la classe `Solaris_LogRecord` est créée dans la base des gestion du CIMOM. Cette instance contient alors une description de l'évènement (origine, date de l'occurence, etc...). Cette information maintenue dans le CIMOM correspond à une entrée dans le fichier de log généré par l'application ;
- `Solaris_LogService` : elle représente l'interface d'accès au service de log, i.e. les méthodes de contrôle et de configuration du service ;
- `Solaris_LogServiceProperties` : elle permet de configurer les fichiers de log utilisés par le service (noms des fichiers, nombre de données par fichier, etc...) ;

9.2.4 Les services XML

Le protocole de communication recommandé par le DMTF pour WBEM s'appuye sur un encodage en XML et un transport HTTP. SUN est l'un des premiers constructeurs à annoncer son support pour ce type de transport. Ainsi, dans la dernière version de son implémentation WBEM, Sun offre un service d'encodage XML pour les données CIM manipulés par l'environnement `Solaris Services`. Elle offre de plus, un service de manipulation d'arbres XML compatible avec le standard DOM (*Document Object Model*).

9.2.5 L'interface de programmation

Distribuée sous le nom de `Sun WBEM SDK`, l'interface de programmation offerte par Sun pour le développement d'applications WBEM est une interface entièrement écrite en Java. Dans sa version actuelle, elle comporte 5 paquetages.

1. *com.sun.wbem.cim* : élément de base pour le développement des applications WBEM, il contient les classes Java permettant de représenter les objets CIM (définition de classe, d'instance, de propriété, etc...) ainsi que les exceptions qui peuvent résulter des erreurs issues des différentes invocations d'opérations CIM ;
2. *com.sun.wbem.client* : contient les classes et interfaces permettant de développer des clients WBEM. La classe **CMIClient**, sans doute la plus intéressante dans ce paquetage, offre des méthodes correspondantes aux opérations CIM définies par le DMTF, permettant ainsi d'interroger la base de gestion d'un CIMOM ;
3. *com.sun.wbem.provider* : contient quatre interfaces `CIMProvider`, `InstanceProvider`, `MethodProvider` et `PropertyProvider` définissant les méthodes à implémenter pour le développement des `providers` ;
4. *com.cun.wbem.provider20* : actuellement, il ne contient qu'une unique définition d'interface `InstanceProvider`, qui represente une mise à jour de l'interface du même nom définie dans le packetage `com.sun.wbem.provider` ;
5. *com.sun.wbem.query* : contient les classes et interfaces utilisables pour créer et envoyer des requêtes au CIMOM. Le langage de requêtes supporté par l'API de Sun est le langage WQL (WBEM Query Langage).

9.2.6 Le schéma Solaris

Parallèlement à la distribution d'une implémentation WBEM, SUN Microsystems a publié un schéma CIM pour la gestion des stations Solaris. Ce schéma s'appuye sur les schéma CIM *Core* et *Common* du DMTF. Dans cette section, nous donnons un bref apperçu sur les composants principaux du schéma `Solaris`.

- `Solaris Core` : le fichier contient les définitions des classes permettant d'implémenter les composants du modèle de base CIM (*Core*). Pour cela il spécialise les classes données par le DMTF pour la gestion du monde Solaris. Les classes les plus importantes définies dans le fichier sont :

  `Solaris_ComputerSystem` : hérite directement de CIM_UnitaryComputerSystem, et permet de modéliser un ordinateur dans un environnement Solaris ;

  `Solaris_LogRecord` et `Solaris_LogService` : ils permettent de gérer les opérations de journalisation ;

- `Solaris Application` : le fichier contient les modélisations des paquetages et des patchs Solaris en CIM. Les classes les plus importantes sont :

  `Solaris_Package` : elle hérite de la classe `CIM_SoftwareElement` du modèle d'applications du DMTF, et modélise un paquetage Solaris. Les attributs de la classe correspondent aux champs de la commande `pkginfo` ;

  `Solaris_Patch` : elle hérite aussi de `CIM_SoftwareElement`, et la spécialise pour modéliser un patch Solaris ;
- `Solaris System` : le fichier contient les modélisations des composants d'un système Solaris, incluant le système d'exploitation et les processus. Les classes les plus importantes du fichier sont :

  `Solaris_OperatingSystem` : elle spécialise la classe `CIM_OperatingSystem` pour l'environnement Solaris ;

  `Solaris_Process` : elle étend la classe `CIM_Process` pour prendre en compte toutes les spécificités des processus dans un environnement Solaris ;
- `Solaris Device` : le fichier contient les spécialisations des définitions du modèle de device du DMTF pour l'environnement Solaris ;
- `Solaris Acl` : il définit les classes Solaris_UserAcl et `Solaris_NamespaceAcl` utilisées par le service de sécurité ;

9.3 Résumé

L'approche WBEM est encore embryonnaire et plusieurs de ses composants ne sont pas encore complètement standardisés par le DMTF (modèle d'évènement, langage de requêtes, etc...). Par voie de conséquence, les deux implémentations restent à ce jour au stade de prototypes expérimentaux, et présentent chacune des avantages et des inconvénients.

L'implémentation de SUN Microsystems est la plus conforme à l'esprit WBEM : une approche unificatrice basée sur les standards approuvés par l'ensemble de la communauté. En effet, cette implémentation présente l'avantage de proposer un protocole de communication HTTP avec un encodage de l'information CIM en XML, standardisé dernièrement au sein DMTF.

D'un autre côté, l'implémentation de Microsoft, bien que nettement moins respectueuse de la norme, présente un avantage de taille : elle comprend un service de gestion des notifications qui n'est pas encore défini par le DMTF, et semble avoir toutes les caractéristiques d'un produit Microsoft.

10 Conclusion

L'approche WBEM en général, et le modèle CIM en particulier, ont rapidement suscité un intérêt fort auprès des industriels, et ce pour plusieurs raisons. D'une part, ses objectifs sont louables: intégrér les approches existantes. D'autre part, son approche pour la modélisation des informations de gestion et le modèle commun déjà définis sont prometteurs. Un autre intérêt repose sur l'architecture de la base d'informations de gestion qui comporte à la fois les objets gérés et les spécifications de ces derniers. Cela n'est fait actuellement que dans l'approche OSI et encore à grand frais via le développement d'une fonction de gestion spécifique.

Cependant, la principale motivation pour cette approche reste sa relative simplicité et le support d'industriels importants tels que Microsoft, SUN et CISCO.

10.1 Avantages et intérêts de l'approche

Les principaux intérêts de l'approche WBEM par rapport aux approches concurrentes sont:

- le choix d'une approche orientée-objets pour la modélisation des ressources.

  Une telle approche également retenue par l'OSI a le grand intérêt de permettre une modélisation claire, compréhensible et de bonne qualité des éléments gérés;
- la possibilité d'intégrer dans le modèle à la fois des définitions de classes d'objets gérés, de relations entre les objets gérés et d'instances d'objets gérés et de fournir un espace de nommage identique pour les deux.

  Ceci n'est pas le cas dans l'approche OSI ou une MIB contient des instances d'objets gérés et dans laquelle on ne peut pas ajouter de nouvelles classes à la demande.

  De plus, CIM ne nécessite pas le développement d'une fonction de gestion pour le partage de l'information de gestion (OSI SMK) car le modèle est intrinsèquement disponible au sein du serveur;

- la liaison avec une méthode de conception orientée-objets (ici UML) et un lien direct entre la représentation graphique et textuelle des modèles de l'information.

  Ceci n'est notamment pas le cas dans GDMO ce qui a certainement freiné longuement son utilisation et a largement favorisé les critiques à l'encontre de la notation;
- l'approche présente une bonne base pour l'intégration des autres approches de gestion avec un modèle de l'information "maniable" (via l'utilisation des qualifieurs) et facile à apprivoiser.
- le protocole de communication présenté s'appuye sur deux standards de WEB très utilisés offrant ainsi un excellent support d'interopérabilité entre applications.

L'intérêt principal de cette approche reste cependant le fantastique support industriel qu'elle suscite dans le monde des fabriquants de PC, dans celui des stations de travail et plus récemment des équipementiers de télécommunications.

10.2 Limites de l'approche

WBEM a également un certain nombre de limites. Parmi ces limites on trouve:

- la notion de qualifieur est intéressante mais retire des classes d'objets beaucoup de leur sémantique. En effet, l'on va mettre énormément d'information dans les qualifieurs pour distinguer les types de classes alors que cela aurait pu se faire en définissant directement au niveau du méta-modèles des classes spécifiques par type d'objets (objet gérés, associations, indications, ..).
- tout comme l'ensemble de ses concurrents, le langage MOF de l'approche CIM ne comporte pas de support pour la description du comportement des objets gérés. Seul un support minimal pour la spécification des conditions de déclenchement de notifications est fourni via le concept de trigger.
- le manque cruel de CIM, tout comme l'approche WBEM en général est l'abscence d'interface de programmation d'application qui permettrait le développement de l'approche WBEM dans l'industrie. Aujourd'hui chaque constructeur fournit une API propre et seul SUN implémente officiellement le support HTTP/XML. Ceci a un effet immédiat : absence totale d'interopérabilité;
- les interactions du type producteur/consommateur ainsi qu'un modèle de notifications fait toujours défaut à l'approche. Ceci est de loin la principale limite de l'approche. En effet, en raison de cette absence, WBEM est pour le moment limité à un mécanisme de gestion par scrutation, connu depuis des lustres comme inefficace.
- l'utilisation des protocoles XML/HTTP est lourde pour l'échange des informations de gestion. Les entreprises qui les ont implantés l'ont compris et ont tous une deuxième alternative pour la communication, propriétaire, et plus performante.

Tout comme le principal intérêt, la principale limite est politique. En effet, le consortium DMTF manque aujourd'hui cruellement d'opérateurs et fournisseurs de services, principaux consommateurs des solutions éventuelles du DMTF notamment dans le domaine des modèles réseau et services (politiques). De plus cette approche se heurte au support actuel de ces derniers (opérateurs + ISP) à l'approche CORBA que WBEM no positionne absolument pas par rapport à son modèle de solution.

10.3 État actuel et futur

Si l'on retrouve le terme WEB dans la définition WBEM, on se rend vite compte lors de l'étude de l'approche que finalement, le WEB n'est pas central mais que, vu l'évolution des besoins et l'aspiration des utilisateurs à faire un maximum d'opérations via le WEB, les technologies du WEB pourraient s'intégrer dans WBEM et fournir un bon support. De plus, en retenant le WEB comme composante de l'initiative WBEM, addresse également les critères de réduction de coût de solution de gestion et de facilité d'utilisation.

En ce qui concerne le modèle de l'information, il commence à se stabiliser sous l'implusion des groupes de travail du DMTF, et les éléments de base (`Core` et `Common`) sont définis, mais il reste encore du travail au niveau de la définition d'un modèle de notifications.

L'approche fournit, certe, un premier pas vers l'interopérabilité et la portabilité des applications de gestion, mais jusqu'à ce jour, l'intégration des approches existantes est restée embryonnaire et limitée à seulement quelques unes parmi elles.

Malgré ces retards et incomplétudes, l'avenir de CIM s'annonce prometteur. En effet, la disponibilité d'environnements de développement conviviaux de la part de SUN, Microsoft et du SNIA comblent d'entrée les

lacunes identifiées il y a quelques années pour la gestion OSI pour laquelle les environnements logiciels faisaient cruellement défaut. De plus, l'utilisation de CIM dans l'approche DEN soutenue par CISCO garantit au modèle un impact fort dans le monde des réseaux et surtout des services, domaines de prédilection de la supervision. Finalement, le fait que des initiatives telles que JMX qui regroupent de nombreux constructeurs de plates-formes de supervision intègrent dès leur première versions de spécification des interfaces WBEM démontre l'impact indéniable de cette approche sur le monde de la supervision.

11 Acronymes

CIM : Common Information Model.
CIMOM : CIM Object Manager.
COM : Component Object Model.
CMIS : Common Management Information Service.
CMIP : Common Management Information Protocol.
DCOM : Distributed Component Object Model.
DEN : Directory-Enabled Networks
DMI : Desktop Management information.
DMTF : Distributed Management Task Force.
GRM : General Relationship Model.
HTTP : Hyper-Text Transport Protocol.
JMX : Java Management eXtensions.
LDAP : Lightweight Directory Access Protocol.
MIB : Management Information Base.
RGT : Réseau de Gestion des Telecommunications.
SNMP : Simple Network Management Protocol.
SNIA : Storage Networking Industry Association.
TMN : Telecommunication Management Network.
UML : Unified Modeling Language.
USB : Universal Serial Bus.
WBEM : Web-Based Enterprise Management.
WMI : Windows Management Interface.
WQL : WBEM Query Langage.
XML : eXtensible Markup Langage.

Références

[1] Microsoft Management Console : Overview. Technical report, Microsoft Corp., 1998. White Paper, http://www.

[2] Windows Management Instrumentation and Simple Network Management Protocol. Technical report, Microsoft Corp., 1998. White Paper, http://www.

[3] Windows Management Instrumentation Scripting. Technical report, Microsoft Corp., 1998. White Paper, http://www.

[4] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A Simple Network Management Protocol, 1990.

[5] DMTF. Desktop Management Interface Specification, version 2.0, June 1998. http://www.dmtf.org/spec/spec.html.

[6] DMTF. XML as a Representation for Management Information, a White Paper. Technical report, Desktop Management Task Force, September 1998.

[7] DMTF. CIM XML DTD, version 2.0. Technical report, Distributed Management Task Force, July 1999.

[8] DMTF. Common Information Model (CIM) version 2.2. Technical report, Distributed Management Task Force, June 1999.

[9] DMTF. Specification for CIM Operations over HTTP, version 1.0. Technical report, Distributed Management Task Force, August 1999.

[10] DMTF. Specification for the Representation of CIM in XML, version 2.0. Technical report, Distributed Management Task Force, July 1999.

[11] DMTF. WBEM Query Langage (Draft). Technical report, Distributed Management Task Force, February 2000.

[12] LaBarre Lee (Editor). Forum026 - Translation of Internet MIBs to ISO/CCITT GDMO MIBs, Issue 1.0, October 1996.

[13] O. Festor, L. Festor, P. Andrey, and Ben Youssef N. Integration of WBEM-based Management Agents in the OSI Framework. page to appear, 1999. IM 99.

[14] O. Festor and N. Ben Youssef. Intégration du modèle de l'information de gestion CIM dans l'approche OSI. Rapport technique RR-3647, INRIA, Mars 1999.

[15] IETF. Hypertext Transfer Protocol – HTTP/1.1. Technical report, IETF RFC 2068, January 1997.

[16] IETF. HTTP Extension Framework. Technical report, IETF Draft, March 1999.

[17] Sun Microsystems Inc. Java Management Extensions, CIM/WBEM APIs. Technical report, Aout 1999.

[18] Sun Microsystems Inc. WBEM on Sun Developper's Guide. Technical report, Aout 1999.

[19] Sun Microsystems Inc. Solaris WBEM Services, Administrators Guide. Technical report, Fevrier 2000.

[20] ISO-8879. Standard Generalized Markup Langage (SGML), October 1986.

[21] ISO-9595. Common Management Information Service Definition, 1991.

[22] S. Judd and J. Strassner. Directory-Enabled Networks. Technical report, Fevrier 1998. Draft.

[23] Microsoft. Windows Management Instrumentation : Background and Overview. Technical report, Microsoft Corp., 1998. White Paper, http://www.ntserver.com/netserver.

[24] Microsoft. Windows Management Instrumentation and the Commmon Information Model. Technical report, Microsoft Corp., 1998. White Paper, http://www.ntserver.com/netserver.

[25] N. Soukouti and U. Hollberg. Joint Inter Domain Management: CORBA, CMIP and SNMP. pages 153–164, 1997. IM97.

[26] W3C. Extensible Markup Langage (XML), version 1.0, December 1997.

[27] M. Wahl, T. Howes, and S. Kille. Lightweight Directory Access Protocol (v3). Technical report, Décembre 1997.

Unit´e de recherche INRIA Lorraine, Technopˆole de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhˆone-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399